UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN MÓVIL PARA GESTIONAR LA COMUNICACIÓN CON ESTUDIANTES DE POSGRADO

Javier de Frutos Gómez
Tutor: Álvaro Ortigosa Juárez

JUNIO 2019

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN MÓVIL PARA GESTIONAR LA COMUNICACIÓN CON ESTUDIANTES DE POSGRADO

AUTOR: Javier de Frutos Gómez
TUTOR: Álvaro Ortigosa Juárez

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2019

Resumen

Vivimos en un mundo en el que cada vez nos encontramos más rodeados de tecnología en todos los ámbitos de la vida cotidiana. La educación no es una excepción y a lo largo de los años la tecnología ha irrumpido dentro de las aulas mediante aplicaciones como Moodle. Además, la información contenida antiguamente en tablones de anuncios ahora es consultable desde cualquier lugar a través de páginas web. Sin embargo, los equipos docentes tienen la impresión de que los alumnos no aprovechan como tanto como debieran todos estos medios.

Desde la Facultad de Ciencias Económicas y Empresariales se ha propuesto una solución a este problema, la creación de una aplicación para teléfonos móviles inteligentes en la cual se reúnan los contenidos informativos de los cursos de máster de dicha facultad. De esta manera se pretende hacer llegar la información importante a los alumnos de la manera más sencilla para ellos y en cualquier lugar.

Desde el punto de vista técnico, esta aplicación resulta ser una gran oportunidad para que el autor se familiarice con las técnicas de desarrollo web más actuales, como son las aplicaciones reactivas basadas en Javascript y el manejo de bases de datos NoSQL. Además, este proyecto pone al alumno en la situación de un consultor que deberá reunirse con el cliente, entender y analizar sus necesidades para plasmarlas en un diseño que las satisfaga, a la vez de ser realista respecto a lo que es posible alcanzar con el tiempo disponible para el desarrollo del sistema.

En este documento se van a recoger las fases del ciclo de desarrollo del proyecto. Primero, se comenzará con una introducción en la se establecerán los objetivos y motivación del proyecto. Se continuará con una descripción del estado del arte, explicando las alternativas disponibles a la aplicación propuesta y explorando las tecnologías disponibles para su desarrollo.

Tras lo anterior, comenzará la fase de análisis, en la que se concretará la descripción del sistema y los requisitos que debe cumplir. A partir de ahí, se llevará a cabo el diseño del sistema desde un punto de vista técnico basado en una aplicación distribuida que utiliza el framework Meteor JS, afrontando su arquitectura distribuida entre cliente y servidor, modelo de datos de la base de datos Mongo DB, interfaces, y estructura de código. En este punto, se pasará a plasmar el diseño en una implementación, explicando el entorno de desarrollo, la implementación de los elementos que forman tanto el cliente, como el servidor y finalizando con el despliegue de la aplicación en un servidor real para su uso por parte de los usuarios.

Una vez finalizado el desarrollo, se expondrán las pruebas realizadas sobre el sistema, así como los resultados en las mismas, lo cual llevará a elaborar una serie de conclusiones y líneas de trabajo futuro.

Abstract

We live in a world surrounded by technology in almost every aspect of our lives. Education is not an exception and in the later years, technology has made its way into the classrooms by applications such as Moodle. In addition, all the information that had to be consulted on news boards is now available in web sites. However, although we have all these means, teachers feel that students are not making the most out of them.

The Faculty of Economic Sciences and Business of the Universidad Autónoma of Madrid has come to a solution. They want to create a mobile application for smartphones for their graduate students in which all the important information about their course is contained. By this mean, they want to make the students able to access the information they need in the easiest way, anywhere they are.

From a technical point of view, this app is a great opportunity for the author to get in touch with the most modern and innovative web development technologies, such as Javascript reactive applications or NoSQL database management. In addition, in this project, the author will assume the role of a consultant who will have meetings with his client. From there, he will have to understand and analyze his client's needs and create a design that satisfies those needs, while considering the development time available.

This document contains every phase of the development cycle of the project. First, there is an introduction to the objectives and motivation of the project. Then, it continues with a study of the currently available solutions for the needs of the client and the state-of-the-art technologies for web development.

In this point the analysis phase will begin, in which a full system description will be performed, followed by the functional and non-functional requisites that the system must fulfill. From there, the design phase starts. This phase will describe the architecture of the application, which is based in the framework Meteor JS, and the database, based in Mongo DB. It will also explore the user interface design and the code structure.

After the design phase, the development phase begins. This chapter will contain all the information about the development environment, the front-end implementation, the back-end implementation and finally, the deployment of the application in a real server and app store.

The next chapter after the development contains all the information about the testing done during and after the development of the project. It also contains the results obtained in the tests. Finally, some conclusions will be made from the development and future work will be stablished.

Palabras clave

Aplicación, Javascript, Reactividad, Multiplataforma, Meteor, MongoDB, NoSQL, Android, IOS, distribuida, servidor, cliente, base de datos, aplicación web, Heroku, Node, pila completa.

Keywords

Application, Javascript, Reactivity, Multiplatform, Meteor, MongoDB, NoSQL, Android, IOS, distributed, server, client, database, web application, Heroku, Node, full-stack.

Agradecimientos

En primer lugar, quiero agradecer a mi tutor, Álvaro Ortigosa, permitirme haber realizado este trabajo, así como asesorarme en lo necesario y emplear su tiempo para acompañarme en las reuniones que tuvieron lugar durante el proyecto con el cliente.

También quiero agradecer a la Facultad de Ciencias Económicas y Empresariales y en especial a José Luis Méndez García de Paredes por su confianza en mí para la realización de este proyecto y su retroalimentación a lo largo del mismo.

Agradezco mucho a mis padres el apoyo que me han proporcionado a lo largo del grado, ayudándome y poniendo todos los medios al alcance de su mano para que avanzase.

Es también necesario agradecer a mis compañeros por todos los momentos y recuerdos que quedan.

Por último, tengo que agradecer a la Escuela Politécnica Superior por haberme permitido madurar mi forma de pensar y desarrollar mis capacidades mientras completaba mi formación.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Este TFG va a tratar sobre el diseño y desarrollo de una aplicación móvil para mejorar la comunicación entre la Facultad de Ciencias Económicas y Empresariales de la Universidad Autónoma de Madrid y sus alumnos de postgrado, creando una plataforma donde la información llegue a los últimos de una manera rápida y sencilla.
Actualmente, la Facultad de Ciencias Económicas y empresariales cuenta con una oferta de seis másteres oficiales distintos. Cada máster cuenta con su propia página web, dentro de la web de la facultad de Ciencias Económicas y Empresariales, en la que se recogen noticias relevantes del máster, eventos, información académica, horarios, matrícula, profesorado, etc.

Las páginas web de los distintos másteres son similares y la información contenida en estas se encuentra bien estructurada y accesible para los alumnos. Sin embargo, el equipo docente de máster de la Facultad de Ciencias Económicas y Empresariales ha tenido a lo largo del tiempo y de manera continuada la impresión de que los alumnos no acceden con regularidad a la respectiva página web de su máster, o que cuando acceden no consultan toda la información que les podría ser relevante. Esto ha llevado a pensar a la dirección del centro que existe la necesidad de crear un canal de información entre el centro y sus alumnos que permita que los últimos encontrar información relevante rápidamente, además de estar al tanto de todo lo que sucede en su facultad y de todas las oportunidades de formación y laborales existentes, ofrecidas por la facultad dentro de su especialidad. Esta información debe llegar a los estudiantes de una manera rápida y sencilla, dado que, si se implementa en un sistema en el que se requiere muchas acciones por parte de los alumnos hasta conseguir la información, estos no utilizarán el sistema con la frecuencia necesaria. Por ello, se ha decidido crear una aplicación para smartphone distribuida en la que se implemente el sistema descrito.

1.2 Objetivos

El objetivo principal del presente TFG es la implementación de una aplicación distribuida para propiciar la comunicación entre la Facultad de Ciencias Económicas y Empresariales y sus alumnos de postgrado. Además, se tiene también como objetivo realizar este desarrollo de una manera lo más parecida, dentro de lo posible, a como se realizaría en un ámbito profesional real. Para alcanzar este objetivo, la facultad de Ciencias Económicas y Empresariales actuará como cliente y el autor de este TFG como consultor que desarrollará la aplicación. La finalidad de esto es conseguir que el alumno comience a entender cómo son las relaciones entre desarrollador y cliente en un entorno más real que el que se ha podido encontrar hasta ahora a lo largo del grado.
Es también un objetivo cumplir los requisitos establecidos en el análisis de requisitos funcionales y no funcionales. Todos los requisitos del sistema se encuentran recogidos en el apartado 3.3 de este documento.
La realización de este TFG permitirá al alumno introducirse en el mundo del desarrollo full-stack y comprender de manera práctica como interactúan los distintos agentes involucrados en una aplicación distribuida. En relación con lo anterior, este Trabajo Final de Grado tiene como objetivo familiarizar al alumno con las tecnologías de desarrollo web más recientes disponibles actualmente en el mercado. Este objetivo facilitará al alumno su posterior incorporación al mercado laboral actual.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Introducción:** En la introducción de esta memoria hablará sobre la motivación de este proyecto y los objetivos que se buscan conseguir con él.
- **Estado del arte:** En este capítulo se comenzará analizando las aplicaciones de apoyo al estudiante universitario disponibles actualmente. A continuación, se estudiarán las tecnologías de desarrollo web más actuales que resultan relevantes para la realización de este proyecto.
- **Análisis:** Este capítulo realizará una descripción detallada de la funcionalidad que debe implementar este proyecto. A partir de esta, se extraerán los requisitos funcionales y no funcionales del sistema. Por último, se describirán las reuniones mantenidas con el cliente.
- **Diseño:** Este es uno de los capítulos más importantes de esta memoria. Aquí se definirá la arquitectura del sistema a implementar, así como la estructura de la base de datos que se utilizará. También se describirá la estructura de archivos del proyecto, para finalizar con el diseño de las interfaces de usuario de la aplicación.
- **Desarrollo:** Este es otro de los capítulos más importantes de este documento. Aquí se plasmará el diseño en la codificación de los distintos componentes del sistema. Se empezará estableciendo el entorno de desarrollo y creando una aplicación y sus dependencias, para pasar al desarrollo de front-end del sistema, describiendo los distintos tipos de elementos que lo forman. Se continuará con el back-end y los elementos que lo forman. Finalmente, se explicará el proceso completo de despliegue de la aplicación.
- **Integración, pruebas y resultados:** En este capítulo se describirán las pruebas realizadas durante y después del desarrollo del sistema, además de los resultados que se han observado en dichas pruebas.
- **Conclusiones y trabajo futuro:** Este capítulo final recoge las conclusiones obtenidas después de realizar este proyecto y las posibles líneas de trabajo futuras para continuar su crecimiento.
- **Referencias:** Lista de la bibliografía consultada para la realización de este proyecto

- **Glosario:** Lista de siglas utilizadas en este documento
- **Anexos:**
    - **Anexo A:** Requisitos funcionales del sistema
    - **Anexo B:** Requisitos no funcionales del sistema
    - **Anexo C:** Lista de rutas de la aplicación
    - **Anexo D:** Lista de métodos de la aplicación
    - **Anexo E:** Diseño de interfaces de usuario de la aplicación móvil
    - **Anexo F:** Diseño de interfaces de usuario de administración

2 Estado del arte

2.1 Alternativas existentes

A continuación, se mostrarán una serie de aplicaciones y sitios web que ofrecen una funcionalidad similar a la aplicación a desarrollar, sin cumplir totalmente las expectativas de la Facultad de Ciencias Económicas y Empresariales. Se van a analizar las características que resultan interesantes, a la vez que sus carencias. Con ello, se podrá alcanzar una visión clara de qué necesidades tiene la nueva aplicación que trata este TFG atender y por qué los sistemas actualmente disponibles no son totalmente válidos.

2.1.1 Página web de máster

En la página web de la Facultad de Ciencias Económicas y Empresariales, se puede encontrar un apartado de postgrado en el cual cada máster dispone de su propia página de información, noticias, evento, plan de estudios, horarios y otros datos sobre el máster.

Figura 1: Página web del Máster en Dirección de Marketing

Sin embargo, este sitio web tiene la desventaja de no ofrecer ninguna interacción entre los alumnos ni facilitar la comunicación entre profesores. Además, desde la Facultad se hace hincapié en que los alumnos no consultan la página todo lo que deberían, puesto que les exige navegar hasta ella y no está adaptada para dispositivos móviles, por lo que una experiencia óptima solo puede ser alcanzada utilizando un ordenador convencional.

2.1.2 Moodle

Moodle es una herramienta de gran difusión entre las universidades que permite a los alumnos acceder a una gran cantidad de información sobre las asignaturas que se encuentren cursando, además de permitirles interactuar entre ellos. Cuenta con una aplicación móvil, aunque está diseñado para ser utilizado principalmente en sobremesa. En el caso de la funcionalidad que se pretende conseguir, Moodle se encuentra principalmente enfocado a

las asignaturas que un alumno cursa, más que al máster que realiza. Además, cuando un alumno finaliza sus estudios, finaliza su relación con Moodle, por lo que no es válido de cara a que los alumni de cursos anteriores se mantengan informados de las últimas novedades de su especialidad.

2.1.3 Alternativas comerciales

Actualmente, existen en el mercado distintas aplicaciones orientadas a fomentar la comunicación entre alumnos y profesores. Sin embargo, muchas de ellas están enfocadas a la Educación Primaria o la Educación Secundaria Obligatoria. Esto conlleva que estas aplicaciones introducen la figura de los padres dentro de la aplicación, cosa que es totalmente innecesaria para los alumnos universitarios. Además, y en línea con lo anterior, la temática de este tipo de aplicaciones suele incluir a personajes animados o dibujos infantiles que no tienen cabida en un entorno adulto como es la universidad. Algunos ejemplos de este tipo de aplicaciones son *ClassDojo*, *Remind* o *SeeSaw*.
Por su parte, Google ofrece su aplicación *Classroom*. Sin embargo, esta está orientada a que los alumnos se apunten a determinadas asignaturas y sus profesores les propongan distintos ejercicios o lecciones.
Todo lo anterior lleva a concluir que no existe aún una alternativa totalmente válida para las necesidades propias de la Facultad de Ciencias Económicas y Empresariales y que este proyecto brinda la posibilidad de elegir las funcionalidades más interesantes para la Facultad y empaquetarlas en un solo sistema fácilmente accesible para los alumnos de postgrado.

2.2 Tecnologías

Una vez hemos establecido la necesidad de implementar una nueva aplicación para fomentar la comunicación entre los alumnos y la Facultad de Ciencias Económicas y Empresariales, que además se ajuste a las necesidades propias de la misma, es momento de analizar las tecnologías disponibles para desarrollar el sistema.

2.2.1 Meteor

Meteor JS es un framework basado en Javascript que permite la programación de aplicaciones para navegador y para móviles. La principal característica de este framework es que es multiplataforma y full-stack, lo cual lo hace muy atractivo de cara al desarrollo de la aplicación de este TFG. Que se trate de un sistema multiplataforma, permite que con único proyecto se puedan atender a las necesidades de ofrecer una aplicación móvil capaz de funcionar tanto en el sistema operativo “Android”, como en el sistema operativo “IOS”. Además, la interfaz de administración de la aplicación, donde los coordinadores de máster añadirán los distintos contenidos de la aplicación, puede ser también programada e integrada dentro del mismo sistema Meteor.
Meteor es un framework que ofrece mucha libertad al desarrollador a la hora de diseñar la aplicación y esto se aprecia en que no sigue ningún modelo concreto, pudiendo ordenar los archivos y la funcionalidad como se considere. Además, incorpora soporte para bases de datos Mongo DB, que se describirán más adelante.

2.2.1 Javascript

Javascript es un lenguaje de programación de alto nivel surgido en 1995 de gran popularidad en el desarrollo web y altamente soportado por todos los navegadores web populares.
Inicialmente fue concebido como un lenguaje para programación de clientes dentro de navegadores a la hora de crear páginas web dinámicas. Sin embargo, ha evolucionado a lo largo de los años para llegar a permitir su uso tanto en el cliente, como en los servidores e incluso en aplicaciones nativas.
Permite el uso de programación imperativa, funcional o por eventos. Además de dar soporte a Objetos, con la desventaja de que el rendimiento de estos objetos no puede llegar a ser excesivamente óptimo. Además, se trata de un lenguaje no tipado, lo que tiene como consecuencia que resulte más difícil de mantener y escalar por parte de terceras personas. La no tipificación también resulta en una mayor complejidad a la hora de depurar el código escrito en este lenguaje.

2.2.2 Node.js

Node.js es un entorno de ejecución para Javascript en el que se basa el framework Meteor. La principal característica de Node es que permite ejecutar código Javascript fuera del navegador. Esto permite que las aplicaciones Meteor estén basadas puramente en Javascript, tanto para el cliente como para el servidor.
Una característica que diferencia a los servidores Node frente a otros lenguajes es que están utilizan programación basada en eventos en lugar de depender de hilos. Por lo que aparece la figura de las promesas en el lado del servidor.
El uso de Node aporta la ventaja de poder acceder a NPM, siglas de *Node Package Manager*. Este sistema de paquetes permite instalar librerías de manera rápida en las aplicaciones, así como satisfacer todas las dependencias que puedan existir en la aplicación de manera automática.

2.2.3 Renderizado de interfaces gráficas

A la hora de desarrollar una aplicación web, es necesario contar con una interfaz de usuario. El framework Meteor incorpora soporte para los frameworks de front-end Blaze, React y Angular.

2.2.3.1 Blaze

Blaze es framework front-end por defecto de Meteor. Es un framework desarrollado por la propia Meteor. Se basa en un sistema de templates HTML5 que, además, pueden utilizar un lenguaje especial, denominado *Spacebars*, que permite la interacción con datos reactivos. Dado que es el framework que cuenta con un mayor nivel de soporte e integración dentro de Meteor, es la opción elegida para este proyecto.

2.2.3.1 React

React es una librearía Javascript destinada a la construcción de interfaces de usuario. Tiene soporte para trabajar con datos reactivos y permite el desarrollo de interfaces de usuario de manera rápida. Sin embargo, su documentación no es tan extensa como otras opciones, por lo que presenta una curva de aprendizaje mayor. Otra desventaja es que fue concebido como una librería para el desarrollo de interfaces de usuario más que como un framework front-end.

2.2.3.1 Angular y Ionic

Angular es una plataforma de código abierto para el desarrollo front-end creada por Google. Es muy popular y cuenta con una documentación completa, lo que facilita su uso. En el caso de Meteor, el soporte a Angular se da a través de Ionic.
Ionic es un framework destinado al desarrollo de aplicaciones móviles multiplataforma, característica que cuadra con los intereses para este proyecto. Sin embargo, finalmente se eligió Blaze en lugar de Ionic debido a que la integración de Meteor con Ionic no es tan limpia en la práctica como cabría desear. Es necesario satisfacer muchas dependencias con versiones de paquetes que no siempre son fáciles de encontrar. De hecho, en proyectos de mayor complejidad pueden existir problemas de compatibilidad entre Meteor y Angular/Ionic debido a esto último. Es por ello por lo que se ha descartado esta alternativa.

2.2.1 Mongo DB

Mongo es un sistema de bases de datos no relacionales o NoSQL. Las bases de datos NoSQL son bases de datos que surgieron a finales de la década de los 90. Se caracterizan por no seguir una estructura fija como es el caso de las bases de datos relacionales.
En el caso de MongoDB, la base de datos se compone de colecciones. Una colección sería el equivalente a una tabla de una base de datos relacional, con la diferencia de que una tabla sigue una estructura y una colección sencillamente almacena documentos. Los documentos se pueden equiparar a las filas de una base de datos relacional. Cada documento en MongoDB es independiente del resto, por lo que puede contener campos totalmente distintos al resto. Una ventaja de esta aproximación es que es muy sencillo actualizar colecciones para que contengan nuevos campos en sus Documentos.
Que no exista una estructura fija no implica que no se puedan realizar consultas. De hecho, las bases de datos NoSQL son reconocidas por ser muy eficientes en las operaciones de consulta. Además, es posible la creación de índices en las bases de datos MongoDB y realizar operaciones equivalentes a *group by* y *left join*, que no es posible en todas las bases de datos NoSQL

3 Análisis

Este capítulo tratará sobre la fase de análisis del proyecto. Se partirá desde el momento en el que se tiene la primera reunión con el cliente de la aplicación, para continuar con una descripción del sistema a implementar, que llevará a establecer todos los requisitos funcionales y no funcionales del proyecto.

3.1 Reunión inicial con el cliente

Dado que este TFG se asemeja a un proyecto real, en el que un cliente acude a un consultor con un problema para que este lo resuelva, en este caso el cliente ha sido la Facultad de Ciencias Económicas Empresariales y el consultor ha sido el autor de este TFG, Javier de Frutos Gómez.
Tras acordar la asignación del TFG con el tutor de este, Álvaro Ortigosa, se procedió a concertar una reunión con el Vicedecano de Postgrado y Formación Continua de la Facultad de Ciencias Económicas y Empresariales, José Luis Méndez García de Paredes. En esta reunión, con la presencia del tutor de este TFG, el cliente explicó el problema y las necesidades de la Facultad al autor del TFG. A partir de los datos obtenidos, se elaboró una descripción del sistema y se establecieron los requisitos funcionales y no funcionales que se han tratado en el anterior apartado. Además, en esta primera reunión se realizaron algunas sugerencias, tanto por parte del tutor, como por parte del autor, al cliente para ayudarle a establecer las funcionalidades necesarias. De esta reunión, el autor del TFG se comprometió a realizar el análisis de la información recogida y a establecer el diseño de la aplicación de cara a la siguiente reunión.

3.2 Descripción del proyecto

En este apartado se procede a realizar una descripción de la funcionalidad y las características que debe ofrecer el proyecto, de acuerdo a la información recogida en la reunión con el cliente. El sistema se divide en una aplicación móvil para Android y IOS, y una aplicación web para la administración del sistema, todo ello apoyado por un servidor y una base de datos MongoDB.

3.2.1 Aplicación móvil

La aplicación debe permitir a los usuarios registrarse. Sin embargo, para que una cuenta de email pueda darse de alta, un usuario de alto nivel debe darla de alta como autorizada y asignarle un grupo que equivale a un máster. De esta forma cada usuario es enrolado en su respectivo grupo sin lugar a confusión. Los usuarios registrados pueden iniciar sesión en la aplicación móvil y dispondrán de un menú sencillo con el que acceder a las distintas secciones de la aplicación. Además, en este menú se mostrará el nombre del máster en el que están matriculados.
El primer apartado al que pueden acceder los usuarios es el de noticias. Aquí se mostrará una lista de las noticias publicadas relacionadas con su grupo ordenadas cronológicamente. Si el usuario pulsa una de las noticias, la noticia se abrirá y podrá ver todo el contenido de dicha noticia.
El siguiente apartado consiste en una agenda de eventos. En ella, los usuarios pueden ver los próximos eventos relacionados con su máster ordenados cronológicamente en una lista.

A continuación, se encuentra la sección de perfiles. Aquí, los alumnos pueden ver una lista con sus compañeros de máster. Si pulsan en alguno de sus compañeros, accederán al perfil de usuario de esa persona, donde podrán ver la información que ese usuario haya compartido con los demás.
La siguiente sección es la de chat. En este caso, esta sección es solo visible para los profesores y los coordinadores de cada máster. En ella, el equipo docente de cada máster podrá mantener una conversación grupal en la que no participarán los alumnos, dado que se determinó que los alumnos habitualmente utilizan otras aplicaciones de mensajería como "WhatsApp" o "Telegram".
Otra sección disponible en la aplicación es la de guías docentes. Aquí, los alumnos encontrarán una lista de las asignaturas que se cursan en su máster. Al pulsar alguna de las asignaturas, se abrirá la guía docente de esa asignatura en formato PDF en el navegador del dispositivo.
Una sección más es el buzón de sugerencias. En esta sección, los alumnos pueden escribir y enviar sugerencias, asociadas automáticamente a su máster, que el coordinador de dicho máster podrá leer más tarde en la sección de administración.
La última sección de la aplicación es la de ajustes. Aquí, los alumnos pueden cambiar su foto de perfil, su contraseña y su información personal. Además, aquí podrán acceder a la declaración de privacidad de la aplicación y cerrar la sesión.
Por último, siempre que el usuario haya iniciado sesión y no se encuentre en el menú, dispondrá de una barra de navegación en la parte inferior de la pantalla para moverse por las distintas secciones de la aplicación.

3.2.2 Interfaz de administración

De manera paralela a la aplicación móvil, el proyecto cuenta con un sitio web de administración donde los coordinadores y los administradores de la aplicación realizarán las gestiones oportunas.
Esta web cuenta con su propia página de inicio de sesión y redireccionará a la página de inicio de estudiantes de la aplicación a cualquier usuario que no sea coordinador o administrador. Los usuarios que inicien sesión se encontrarán con un menú con los distintos apartados de administración. Además, la interfaz de administración cuenta con un menú de navegación cuando el usuario se encuentre dentro de alguna sección.
En la interfaz de administración se pueden encontrar secciones paralelas a las existentes en la aplicación móvil, en las que los coordinadores pueden añadir o modificar el contenido relacionado con su respectivo máster, ya sean noticias, eventos de la agenda, guías docentes, o la información de los alumnos de ese máster. Además, cuentan con una sección para visualizar el buzón de sugerencias de su máster y otra para añadir emails autorizados a registrarse, que serán asignados a su propio máster.
Por su parte, los administradores de la aplicación podrán ver y modificar los perfiles de usuario de todos los usuarios registrados, añadir emails autorizados a registrarse y, además, contarán con una sección en la que podrán crear nuevos grupos de usuarios en la aplicación.

3.3 Análisis de requisitos

Una vez establecida la descripción completa del sistema, se procedió a extraer los requisitos funcionales y no funcionales del proyecto de este TFG. La lista de requisitos funcionales del sistema se puede consultar en el **Anexo A** de este documento. En el caso de la lista de requisitos no funcionales del sistema, esta se encuentra contenida en el **Anexo B** del presente documento.

4 Diseño

4.1 Arquitectura del sistema

En este proyecto se va a implementar una aplicación distribuida conforme al framework Meteor. La elección de Meteor se debe a que es un framework de desarrollo multiplataforma que además aúna las tecnologías más modernas y populares en cuanto a desarrollo web a fecha de la redacción de este documento. Las aplicaciones distribuidas Meteor cuentan con tres componentes principales: el cliente, el servidor y la base de datos.
Meteor es un framework muy flexible a la hora de diseñar la arquitectura del sistema que se implemente. En el caso de este proyecto, se va a utilizar una aproximación al modelo vista controlador. La estructura de este esquema funciona de la siguiente forma:

- El **modelo** representa los datos almacenados de manera estructurada en el sistema. En este caso, el papel del modelo viene representado por la base de datos Mongo DB.
- La **vista** consiste en las interfaces que se muestran a los usuarios. En ella se muestra la información del modelo, con la que los usuarios interactúan y solicitan cambios. En este proyecto, la vista se ve materializada en las interfaces de usuario de la aplicación móvil y la interfaz de administración.
- El **controlador** consiste en la lógica que comunica la vista y el modelo, transmitiendo las solicitudes al modelo y devolviendo los resultados a la vista. En este proyecto, el controlador consiste en el código Javascript que se encuentra tanto en el lado del cliente, como en el lado del servidor.

El sistema se compone de los agentes incluidos en la **Figura 2**:

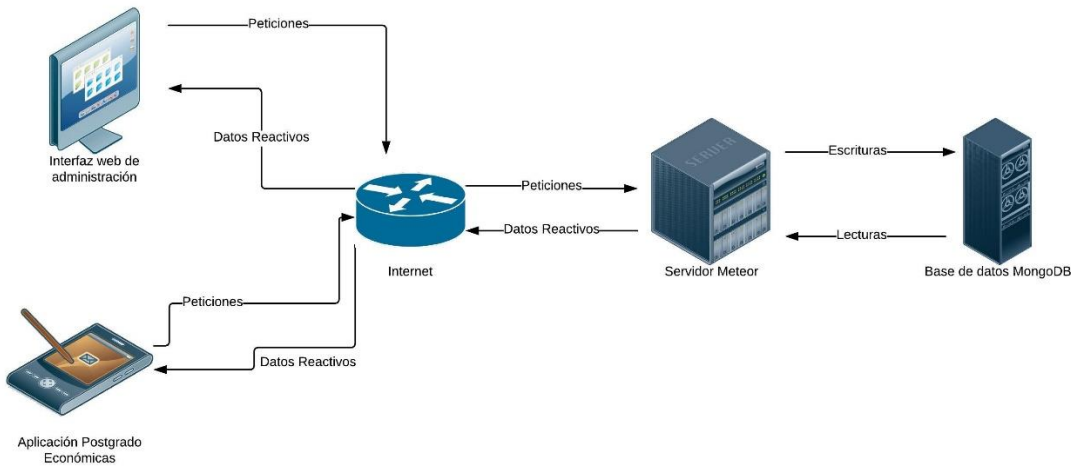

Figura 2: Esquema de la arquitectura del sistema

4.1.1 Comunicación con la base de datos

En este proyecto Meteor hay que entender que esta arquitectura es reactiva y se basa en publicaciones y subscripciones. Esto significa que el servidor es el único que accede directamente a la base de datos y genera publicaciones para cada colección. En este punto, los clientes que deseen acceder a información contenida a la base de datos se subscriben a

las publicaciones que sean necesarias para el controlador de la página en cuestión. Entonces, el servidor sirve la información a dicha subscripción a través de la publicación. Esto tiene dos ventajas. La primera, es que es sencillo filtrar la información que se sirve a través de una publicación de acuerdo a quién sea el cliente que se ha subscrito a ella. La segunda es que permite tener reactividad. Esto es, cada vez que hay un cambio en la base de datos, el servidor sirve la última versión de la información actualizada al cliente, de tal forma que los clientes reaccionan a cambios en el sistema de manera automática.
A la hora de realizar escrituras en la base de datos, el controlador de la página recoge los datos o las órdenes introducidas por el usuario en la vista y las manda al servidor mediante la llamada a un método. En una aplicación Meteor, la ejecución de un método se realiza dos veces. La primera se produce en el lado del cliente, el cuál ejecuta el método con la información que tiene disponible en ese momento y genera un resultado simulado que muestra al usuario mientras recibe el resultado real desde el servidor. El servidor es el segundo en ejecutar el método cuando recibe la llamada desde el cliente. Esta vez, la ejecución del método sí que se realiza con datos reales de la base de datos y permite obtener un resultado real, que puede ser enviado al cliente para su uso.

4.2 Estructura de la base de datos

El modelo de datos que se va a utilizar en este proyecto se va a basar en una base de datos NoSQL MongoDB, la cual tiene integración nativa con el framework Meteor. MongoDB permite que la base de datos sea muy flexible, ya que se permite crear colecciones de datos que cuentan con parejas de clave y valor, dejando al desarrollador la libertad de elegir si se quiere que estas colecciones sigan una estructura o no.
En el caso del sistema de este proyecto, se han estructurado los datos de las distintas colecciones que lo conforman de manera lógica, con las mismas claves para todos los elementos de cada colección, creando así una base de datos bien organizada, fácil de actualizar, pero manteniendo las capacidades de fácil expansión horizontal de las bases de datos NoSQL, de cara a futuras iteraciones de la aplicación.
Para satisfacer los requisitos funcionales del sistema, se han creado una serie de colecciones en Mongo DB que se resumen en la **Figura 3**.

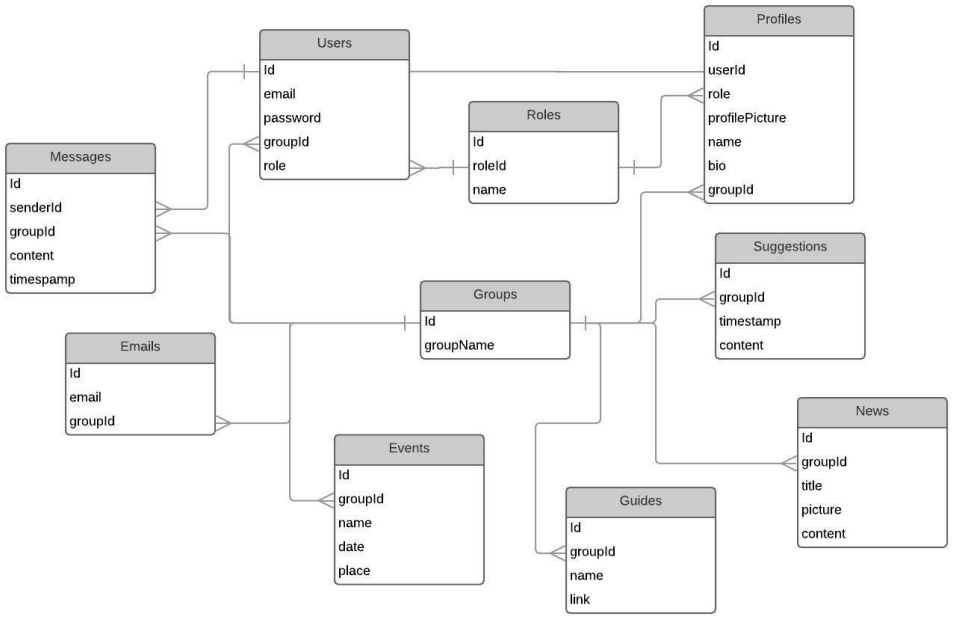

Figura 3: Diagrama de la Base de Datos

En el diagrama de la **Figura 3** se puede apreciar que el eje central de la base de datos es la colección *Groups*, la cual sirve para clasificar la información contenida en el resto de las colecciones a la hora de ser mostrada al usuario.
Por otro lado, es destacable la separación de la colección de usuarios y la colección de perfiles de usuario. Esta decisión se debe a que, para el correcto funcionamiento de la aplicación, es necesario conocer el grupo al que pertenece el usuario o su rol, pero no el resto de la información de su perfil o viceversa. Por ello, para evitar subscripciones excesivas y disminuir la cantidad de datos transferidos entre cliente y servidor se ha optado por separar al usuario y a su información personal en dos colecciones distintas. La utilidad de esta separación se observará en más profundidad en el apartado de desarrollo de este documento.
A continuación, se procede a explicar las distintas colecciones de la base de datos.

4.2.1 Groups

Esta colección almacena los distintos grupos existentes en el sistema, que comprenderán los distintos másteres ofrecidos por la Facultad de Ciencias Económicas y Empresariales, así como a los alumni de postgrado. El único campo necesario, aparte de la Id del grupo, es un *string* que contenga el nombre del grupo. Debido a la dependencia de las demás colecciones hacia los grupos, no se permite a los usuarios eliminar grupos desde la interfaz de administración del sistema.

4.2.2 Roles

En la colección "Roles" se almacenan los roles disponibles en el sistema. Esta colección contiene la Id, el nombre del rol y roleId. RoleId es un valor de 0 a 3 que representa a los 4 roles de la aplicación. El uso de este campo tiene dos ventajas. La primera consiste en que cada vez que la aplicación quiera comprobar un rol, lo hará comparando un valor numérico en lugar de la Id, que es un string, disminuyendo la cantidad de cómputo necesaria. La segunda ventaja, es que son valores que no cambian a lo largo del tiempo, por lo que, aunque se borrase una de las entradas de la colección Roles y se volviese a crear, la Id de ese rol cambiaría, pero no su roleId.
Los roleId disponibles son:

- 0: Estudiante
- 1: Profesor
- 2: Coordinador
- 3: Administrador

4.2.3 Users

La colección Users es creada automáticamente al utilizar el paquete de autenticación de usuarios *Meteor-accounts*. Esta contiene la información de inicio de sesión en la aplicación: email y contraseña. Además, dispone de un campo *profile* donde el desarrollador puede almacenar los campos personalizados que considere oportunos. En el caso de este proyecto, se almacena la Id del grupo al que pertenece el usuario y su rol de acuerdo a un roleId de los explicados en el apartado anterior.

4.2.4 Profiles

La colección Profiles contiene la información personal de los usuarios. Se separa de la colección Users para disminuir la carga de datos entre el cliente y el servidor y disminuir las subscripciones a publicaciones dentro del servidor. En ella se almacena la Id del usuario al que está asociado el perfil, la Id del grupo del usuario, la foto de perfil, el nombre del usuario, su rol y la biografía que el usuario haya escrito en su perfil.
Cabe destacar que dado que el rol y la Id del grupo se almacenan tanto en la colección Users, como en Profiles, los métodos que modifican estos valores para los usuarios lo hacen simultáneamente en ambas colecciones. De esta manera se garantiza la coherencia del contenido de la base de datos.
Por otro lado, la foto de perfil se almacena como un string codificado en *Base64*. Esto resulta extremadamente útil a la hora de representar la imagen en el cliente sin necesidad de aumentar la necesidad de procesamiento.

4.2.5 News

En la colección News se almacenan todas las noticias publicadas en el sistema. El campo Id identifica a la noticia y es el utilizado a la hora de navegar hacia una noticia en concreto. El campo groupId diferencia a cada noticia por el grupo al que pertenece, de cara a mostrarla a cierto usuario o no. Por último, se almacena el título, texto de la noticia, timestamp y la imagen asociada. Igual que en Profiles, la imagen se almacena en *Base64* para facilitar su posterior representado.

4.2.6 Events

La colección Events almacena la información relacionada con los eventos de la sección de agenda de la aplicación. Cada evento se encuentra asociado a un groupId que determina a qué usuarios se les muestra dicho evento, el nombre del evento y su ubicación, ambos almacenados en forma de string y, por último, la fecha del evento, almacenada en formato Date.

4.2.7 Messages

La colección Messages contiene los mensajes enviados en el chat de la aplicación. Además de su propia Id, almacena la Id del grupo al que pertenecen y la Id del usuario que envió el mensaje. Por otro lado, se almacena un string con el contenido del mensaje y el *timestamp* del mensaje en formato Date.

4.2.8 Guides

En la colección Guides se encuentran los enlaces a las guías docentes de las asignaturas que se pueden consultar en el apartado “Guías Docentes” de la aplicación. Para su correcta clasificación, es necesario almacenar la Id del grupo al que pertenecen, así como un string que contiene el nombre de la asignatura y otro string con el enlace de la guía docente en la página web de la Facultad de Ciencias Económicas y Empresariales.

4.2.9 Suggestions

La colección Suggestions almacena todas las sugerencias enviadas a través del buzón de sugerencias de la aplicación. En ella, por cada sugerencia se almacena la Id de la sugerencia, la Id del grupo al que pertenece la sugerencia, un string con el texto de la sugerencia y, finalmente, el timestamp de la sugerencia en formato Date.
Dado que en los requisitos funcionales se establece que las sugerencias deben ser anónimas, no se almacena la Id del usuario que escribió la sugerencia en esta colección y en ninguna otra.

4.3 Diseño de la estructura de ficheros de código

El framework Meteor JS una gran libertad al desarrollador a la hora de estructurar el código, pudiendo, teóricamente, no seguir ningún orden o estructura concreta. Sin embargo, en el caso de este TFG se ha utilizado una aproximación cercana al tradicional modelo vista controlador, de cara a seguir una estructura de organización del código lógica y que facilite tanto el desarrollo, como el posterior mantenimiento del sistema.
La estructura de archivos es la que se observa en la **Figura 4**:

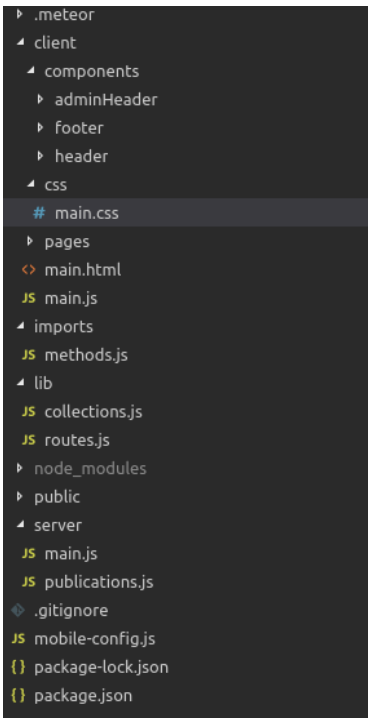

Figura 4: Estructura de ficheros del proyecto

A la hora de diseñar la estructura de ficheros en un proyecto Meteor, hay que tener en cuenta varios factores para conseguir un correcto funcionamiento.
Lo primero que hay que tener en cuenta es que todo lo que se encuentra bajo la carpeta *client*, es código que solo se ejecuta en el cliente, por lo que es aquí donde se colocarán las vistas y controladores de las distintas páginas, componentes de la UI y archivos de estilo. Además, en este lugar se localiza el archivo main.html y main.js del cliente, que sirven de base para la ejecución de la aplicación en el cliente.
Dentro de *client*, las vistas y controladores se localizan dentro de la carpeta *pages*, existiendo una carpeta por cada página, que contiene el código fuente HTML y Javascript, como se observa en la **Figura 5**.

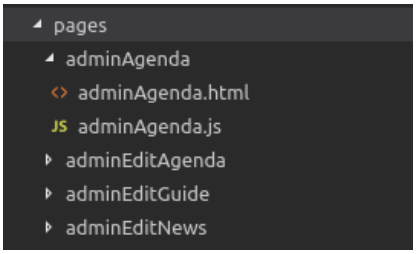

Figura 5: Estructura de ficheros para una página de la aplicación

Lo segundo es que todo lo que se encuentre bajo la carpeta *server* sólo se ejecutará en el servidor. Aquí se encuentran el archivo main.js del servidor y el archivo de publicaciones de las colecciones, a las que se subscribirán los clientes para acceder a la información de la base de datos.

Otro aspecto importante es que el contenido de la carpeta *lib* es lo primero que se carga al ejecutar la aplicación Meteor. Es por ello por lo que dentro de esta carpeta se sitúan el archivo *collections.js* que contiene la definición de las colecciones de la base de datos y el archivo *routes.js*, que contiene las rutas de la aplicación.
La carpeta *imports* puede ser utilizada tanto por el cliente como por el servidor y en ella se incluye el archivo *methods.js*, en el cual se encuentran los métodos que dan soporte a las operaciones de escritura en la base de datos y que son llamados desde el cliente. Es importante que este archivo sea accesible tanto para el cliente como para el servidor porque, cuando se realiza la llamada a un método, el cliente lo ejecuta primero con los datos de los que dispone, simulando una respuesta provisional que muestra al usuario, mientras el servidor recibe la solicitud, procesa el método con datos reales y manda el resultado real al cliente.
Por último, en la raíz de la aplicación, se sitúan los archivos de configuración *mobile-config.js*, que contiene distintas configuraciones para el funcionamiento de la aplicación en dispositivos móviles, y los archivos de paquetes, que recogen las dependencias de la aplicación en paquetes de funcionalidad Meteor.

4.4 Diseño de interfaces de usuario

Dado que el producto final de este proyecto es una aplicación móvil, la interfaz de usuario juega un papel fundamental para el éxito de esta. Tal y como se recoge en los requisitos no funcionales **RNF-9** y **RNF-10**, la aplicación debe tener un diseño atractivo y actual, a la vez que resultar muy sencilla de utilizar por personas no habituadas a realizar gestiones avanzadas con equipos informáticos. En base a estos requisitos se ha optado por una interfaz basada en Bootstrap con un estilo *Material Design*. Bootstrap permite organizar los elementos de la interfaz de manera limpia y clara para el usuario dentro de las plantillas HTML. Añadir estilos *Material Design* a esas plantillas, consigue que los usuarios se encuentren en un entorno familiar, con botones y menús sencillos de navegar y sencillos de localizar sin necesidad de aprendizaje.
En el **Anexo E** se encuentra un catálogo de todas las interfaces de usuario de la aplicación móvil de este proyecto, con una explicación breve de la funcionalidad implementada en cada una de ellas.
El **Anexo F** contiene el catálogo de interfaces de usuario de la interfaz de administración, así como la explicación de la funcionalidad que aportan las mismas.

5 Desarrollo

En este apartado se va a explicar el proceso de desarrollo que se ha seguido en el proyecto, incluyendo la metodología empleada, las tecnologías y los entornos utilizados. Además, se va a concretar el procedimiento empleado para conseguir implementar algunas de las funcionalidades del sistema, de acuerdo a los paradigmas del diseño establecido en el capítulo anterior.

5.1 Tecnologías y lenguajes empleados

Para desarrollar este proyecto se va a utilizar el Framework Meteor tal y como se ha descrito en el apartado de diseño, acompañado del framework de front-end *Blaze* y la base de datos *MongoDB*. El uso de estas tecnologías determina los lenguajes de programación necesarios. Para la creación de las vistas de la aplicación se utilizará el lenguaje *HTML*, al que se añadirá la sintaxis propia de *Blaze*, que, como se verá más adelante, utiliza los caracteres de escape {{}}. El estilo de las vistas vendrá dado por *CSS*, mientras que toda la lógica, tanto del cliente, como del servidor, será desarrollada en el lenguaje Javascript. Finalmente, para el renderizado de las interfaces de usuario en los clientes móviles se utiliza *Apache Cordova*, el cual se basa en node.js.

5.2 Entorno de desarrollo

Para desarrollar un proyecto Meteor es posible utilizar los sistemas operativos *Windows*, *Linux* o *Mac OS*. Se ha elegido *Linux Ubuntu* dado que *Meteor* para *Linux* incluye soporte para desarrollo de aplicaciones móviles, cosa que Meteor para *Windows* no incluye en la versión actual. Para generar la aplicación compatible con dispositivos *IOS* se utilizará un ordenador *Mac OS*, pues así es requerido por *Meteor*.
A la hora de desarrollar en Linux Ubuntu, el editor de texto empleado ha sido *Microsoft VS Code*. Está elección se ha realizado por gusto personal del autor y cualquier otro editor de texto hubiese sido válido.
Para el testeo del sistema se ha utilizado el navegador *Google Chrome* en su versión más actual, así como el emulador Android incluido dentro del IDE *Android Studio*. El uso de *Android Studio* implica también el uso de *Java SDK*.
El control de versiones se ha realizado mediante un repositorio *Git* proporcionado por el servicio gratuito *BitBucket*.

5.3 Creación de la aplicación Meteor

Para poder empezar el desarrollo de la aplicación, es necesario crear la misma y añadir los distintos paquetes, dependencias y plataformas en las que se desee que la aplicación se ejecute. Se trata de una serie de pasos comunes a todos los desarrollos Meteor, que suponen los cimientos sobre los que se construye la aplicación.

5.3.1 Creación de la aplicación Meteor

El primer paso es crear la aplicación. Las aplicaciones Meteor se crean por terminal de comandos. Para ello, se ha de navegar hasta la carpeta deseada e introducir el siguiente comando:

```
meteor create <<nombre de la aplicación>>
```

Este comando crea una carpeta en la cual está contenida la aplicación. Si se navega a ella se pueden añadir las plataformas deseadas para la aplicación. Para añadir *Android*, se utiliza el comando:

```
meteor add-platform android
```

En el caso de *IOS* se utiliza:

```
meteor add-platform ios
```

Nótese que las plataformas pueden ser añadidas o eliminadas en cualquier momento. Para ejecutar la aplicación se dispone de varias opciones. La primera es simplemente ejecutarla en el navegador. Para ello se deberá escribir el comando:

```
meteor
```

En caso de que se desee ejecutar la aplicación en el emulador de *Android*, se deberá utilizar el comando:

```
meteor run android
```

Este comando, además, sirve la aplicación en localhost, por lo que es posible ejecutarla en el navegador simultáneamente.

5.3.2 Paquetes Meteor

Para la implementación de ciertas funcionalidades se han utilizado algunas librerías comprendidas dentro de paquetes Meteor. La instalación de un paquete se realiza mediante el comando:

```
meteor add <<nombre del paquete>>
```

A continuación, se van a explicar los paquetes Meteor utilizados.

5.3.2.1 Blaze-html-templates

Este paquete contiene todas las librerías y dependencias necesarias para poder ejecutar el framework de front-end *Blaze*. Se trata de un meta paquete que se compone a su vez de tres paquetes. El primero se denomina *Templating*, el cual es el compilador de las plantillas utilizadas por *Blaze* ubicadas en archivos *HTML*. El segundo se trata de *BlazeJS*, la librería *Javascript* de *Blaze*. Por último, el tercer paquete es *Spacebars*, el cual es el lenguaje utilizado por *Blaze* para implementar las plantillas.

5.3.2.2 Kadira:flow-router

Kadira:Flow-router es un paquete que aporta funcionalidad de ruteador a la aplicación. Para su uso es necesario definir un archivo de rutas en la aplicación. A partir de ahí, dispone de distintas API para realizar la navegación, añadir parámetros a las URL, o averiguar la ruta actual.

5.3.2.3 Kadira:blaze-layout

Este paquete funciona como complemento de *kadira:flow-router* y permite, como se verá más adelante, introducir el concepto de *templates* de *Blaze* dentro de las rutas de la aplicación, aportando mucha flexibilidad a la hora de combinar las distintas templates disponibles.

5.3.2.4 Accounts-password

Meteor incluye por defecto el paquete "Accounts-base", que suministra soporte al sistema de cuentas de usuario. La adición de este paquete añade soporte al uso de contraseña y da acceso a una serie de APIs muy útiles de cara a gestionar el registro, inicio de sesión o cambio de contraseña de los usuarios de una manera segura.

5.3.2.1 Twbs:bootstrap

El paquete *twbs:bootstrap* permite usar dentro de las aplicaciones Meteor la conocida librería Bootstrap. Esta es utilizada para el desarrollo del front-end, facilitando el desarrollo de interfaces de usuario organizadas y capaz de adaptarse a distintos tamaños de pantalla y navegador. Se basa en un sistema de *grid*, para el cual se incluye una gran cantidad de clases que permiten la organización tanto vertical en filas, como horizontal en columnas de los elementos incluidos en el código HTML de las plantillas. Además, también incluye diversos catálogos de botones, iconos, listas, etc. Que facilitan conseguir un aspecto actual y atractivo en las interfaces de usuario.

5.3.2.2 Momentjs:moment

Este paquete sirve para facilitar la manipulación de fechas en Javascript. Permite *parsear*, modificar y mostrar fechas de manera rápida con las distintas APIs incluidas.

5.3.2.3 Check

Check es un paquete Meteor que sirve para controlar que el valor almacenado en cierta variable se corresponda con lo esperado. Esto es especialmente útil en el lado del servidor, para comprobar que los parámetros que se reciben desde el cliente son correctos, lanzando una excepción y deteniéndola ejecución de, por ejemplo, un método si se detecta que se están recibiendo valores no esperados.

5.3.2.4 Tsega:bootstrap3-datetimepicker

Este sencillo paquete permite utilizar la clase *datepicker* de *Bootstrap*, que no es compatible de forma nativa con *Meteor*. Gracias a esto, la interacción de los usuarios con los formularios mejora drásticamente a la hora de introducir fechas.

5.3.2.5 Session

Session introduce en el cliente la variable reactiva *Session*. Actúa como un diccionario y en este proyecto ha sido especialmente utilizado a la hora de detectar cambios en la interfaz de usuario o a la hora de mostrar u ocultar alguna parte de la interfaz a ciertos usuarios.

5.3.2.1 Cordova-plugin-file

Este paquete es un plugin para el framework *Cordova* que aporta acceso al almacenamiento en dispositivos móviles. La utilidad de este plugin en este TFG reside en dar a los usuarios la habilidad de modificar su foto de perfil cargando una imagen desde la biblioteca de su dispositivo.

5.4 Front-end

El front-end de este proyecto se compone de una aplicación móvil y una aplicación web de administración. En este capítulo se va a analizar las técnicas empleadas para implementar el lado del cliente de este proyecto.

5.4.1 Templates

Meteor Blaze se basa en un sistema de *templates*, plantillas en castellano, para implementar las vistas con las que trabaja el usuario. Una *template* no es más que un archivo HTML que define una serie de elementos que serán renderizados en una página. Cada *template* lleva asociado su correspondiente archivo *Javascript*, con el que se puede conseguir que una *template* sea dinámica y reactiva. Sin embargo, lo más interesante de *Blaze* es que las *templates* pueden ser anidadas. Esto permite crear un concepto de modularidad en el que se utilizan una serie de cimientos comunes para construir distintas páginas en función de lo que se desee mostrar.
En este proyecto se utiliza una *template* maestra denominada *Core*. Esta, se encuentra definida en el archivo *main.html*, que sirve de entrada a la aplicación y su definición es la contenida en la **Figura 6**:

```
<template name="Core">
    <header>
        {{> Template.dynamic template=header}}
    </header>
    <main>
        {{> Template.dynamic template=main}}
    </main>
    <footer>
        {{> Template.dynamic template=footer}}
    </footer>
</template>
```

Figura 6: Template Core

Se puede observar que su diseño es muy sencillo, dividido en las tres secciones habituales de un archivo HTML. Dentro de cada sección se contiene una posición para una *template* dinámica. Por ejemplo, en la posición *main* de *Core* se podrá poner la *template* correspondiente a la página de noticias, de agenda, de chat, etc.

Esta estrategia permite, además, una gran capacidad de expansión de las funcionalidades del sistema. De hecho, tanto la aplicación móvil como la aplicación web de administración están basadas en la template *Core*.
Por último, en la **Figura 7** se muestra una template más convencional. Concretamente, la dedicada al *header* de la aplicación móvil:

```
<template name="Header">
    <div class="header-bar">
        <div class="header-content">
            <h4 class="app-title bold">{{title}}</h4>
        </div>
    </div>
</template>
```

Figura 7: Template de header

Se puede apreciar que dentro del elemento <*template*> el contenido consiste en HTML más tradicional, a excepción de la sintaxis *{{title}}*, que será explicada en la sección *Reactividad*. El resto de templates del proyecto siguen una estructura parecida a este último ejemplo, adaptada a las necesidades de la página en cuestión.

5.4.2 Rutas y navegación

Toda aplicación web necesita de algún sistema de ruteo. En el caso de este proyecto, las rutas se encuentran definidas en el archivo *lib/routes.js* y se basan en los paquetes *Flowrouter* y *Blaze layout*, explicados en la sección *Paquetes Meteor*. El primero aporta la navegación, el segundo incluye a la navegación el sistema de *templates* dinámicas. Las definiciones de las rutas del proyecto son similares entre sí y siguen el esquema de la **Figura 8**.

```
FlowRouter.route('/news', {
    name: 'Noticias',
    triggersEnter: [function (context, redirect) {
        if(!Meteor.userId()){
            redirect('/login');
        }
    }],
    action() {
        BlazeLayout.render("Core",{
            main: "News",
            footer: "Footer"
        });
    }
});
```

Figura 8: Definición de una ruta

Como se observa en la figura, la definición de una ruta se trata de una llamada a la API *FlowRouter.route()*. En ella el primer argumento es la dirección de la ruta y el segundo, la definición del nombre de la ruta, el disparador a ejecutar al entrar a esa ruta y la acción que conlleva esa ruta. En el ejemplo, la ruta es */news*, su nombre es *Noticias*, el disparador comprueba el usuario haya iniciado sesión, y en caso negativo redirigirle a la página de inicio de sesión, y la acción es renderizar la template *Core*, incluyendo dentro de ella las templates necesarias para mostrar la sección de noticias de la aplicación móvil.

Cabe destacar que el disparador resulta muy útil para gestionar permisos de acceso a distintas partes de la aplicación y que en la acción es el lugar donde se rellenan las posiciones de la *template* maestra *Core*. Es también importante tener en cuenta que no es necesario rellenar todas las posiciones de la *template* maestra. En el ejemplo la posición *header* se omite y solo se rellenan *main* y *footer*. El efecto de esto es sencillamente que no se renderiza nada antes de *main*, sin dejar espacios vacíos desagradables para el usuario.

En el **Anexo C** se incluye una lista de todas las rutas disponibles en la aplicación.

5.4.2.1 Enlaces

Para acceder alguna de las rutas descritas en el apartado anterior, es necesario dar una orden de navegación a la aplicación. Esto se puede realizar de dos formas.

- El primer método para enlazar con otra página es sencillamente utilizar un elemento enlace HTML e introducir en el campo *href* la dirección deseada, por ejemplo */settings*.
- El segundo método es utilizar *FlowRouter* el cual incorpora el método *Flowrouter.go("Enlace")*, donde enlace equivale a la dirección del enlace deseado, por ejemplo */settings*.

En cualquiera de los dos casos la aplicación buscará la ruta adecuada para el enlace activado y renderizará la página en función de lo establecido en la declaración de dicha ruta.

5.4.3 Subscripciones

Los clientes que deseen obtener información de la base de datos deberán hacerlo a través de subscripciones. Aunque es posible acceder directamente a la base de datos desde el cliente, este método no garantiza la seguridad del sistema y aumenta el *payload* que se mueve entre cliente y servidor. Es por ello por lo que las subscripciones aportar una manera segura y reactiva de obtener la información necesaria para presentársela al usuario.
El primer paso para realizar una subscripción es disponer de una colección y una publicación en el servidor. Este procedimiento se describirá en detalle el capítulo *Back-end*. De cara al cliente, una vez se dispone de la colección y publicación necesarias, se procede a importar la colección y a subscribirse a la publicación en el momento de crear la página que va a consumir los datos, llamando a la API:

```
Meteor.subscribe(nombre_subscripcion, <<argumentos>>)
```

Esta acción se realiza en el controlador, tal y como se muestra en la **Figura 9.**

```
import { News } from '../../../lib/collections.js'
import './news.html';

Template.News.onCreated(function newsOnCreated() {
    Meteor.subscribe('news', Meteor.userId());
});
```

Figura 9: Creación de una subscripción

Una vez realizada la subscripción, los datos comenzarán a llegar al cliente pasados unos milisegundos. Es importante tener esto en cuenta, de cara a no intentar acceder a datos que no pueden estar disponibles aún debido a que la subscripción todavía no está lista.
Para leer los datos de la subscripción, basta con utilizar los métodos *find()* para obtener una lista de documentos, o *findOne()* para obtener un documento de la subscripción.

```
news() {
    const news = News.find().fetch();
    return news;
}
```

Figura 10: Ejemplo de uso de la función find()

5.4.4 Llamadas a métodos

Cuando los usuarios rellenen formularios o realicen otras acciones en el sistema que generen datos, estos deben ser almacenados en la base de datos. Una aproximación a este problema podría ser que los datos se almacenasen directamente desde el cliente. Sin embargo, esto acarrearía grandes problemas de seguridad. Es por ello por lo que surge la necesidad de utilizar métodos a la hora de guardar la información en la base de datos.
Para realizar la llamada a un método, basta con utilizar la API *Meteor.call()* de la manera que se muestra en la **Figura 11**.

```
Meteor.call('editGuide', guideId, subjectName, url);
```

Figura 11: Ejemplo de llamada a un método

El primer parámetro de *call* será siempre el nombre del método. Los parámetros restantes son los parámetros de entrada del método en cuestión.
En el capítulo sobre el back-end se describirá cómo funcionan los métodos desde el lado del servidor. En cuanto al lado del cliente, dado los métodos se encuentran en el fichero *imports/methods.js*, son visibles para el mismo, por lo que mientras se espera a que el servidor atienda la llamada, el cliente ejecuta el método con los datos de los que dispone a través de las subscripciones y genera un resultado simulado de la ejecución del método. El resultado real llegará a través de las subscripciones cuando se finalice la ejecución en el servidor.

5.4.5 Reactividad

Como se ha explicado en el apartado de diseño de este documento, este proyecto implementa una aplicación reactiva. En la sección anterior se ha expuesto cómo obtener un flujo de datos reactivos en el cliente y en el capítulo *Back-end* se expondrá cómo ofrecer ese flujo de datos al cliente. En esta sección se va a exponer cómo trabaja el cliente con datos reactivos.

5.4.5.1 Helpers

Los *helpers* son métodos que se ubican en el código *Javascript* del front-end de cada página de la aplicación. Estos métodos son llamados por la vista a través de la sintaxis de *Spacebars*, que se explicará en la siguiente sección. En la mayoría de los casos, su función es recuperar información de la base de datos y procesarla para ser representada en la vista, así como decidir si ciertos elementos deben ser representados.

```
Template.Agenda.helpers({

    event() {
        const events = Events.find().fetch();
        for (i in events){
            events[i].time = moment(events[i].date).format('HH:mm');
            events[i].date = moment(events[i].date).format('DD/MM/YYYY');
        }

        return events;
    }
});
```

Figura 12: Ejemplo de helper

En la **Figura 12** se puede observar un ejemplo de un *helper* que recupera una lista de eventos de la base de datos, formatea las fechas de los eventos y los devuelve a la vista, la cual los representará en la página. En la siguiente sección se verá cómo representar esos datos en la vista.

5.4.5.2 Templates

Las *templates Blaze* descritas anteriormente en este capítulo cuentan con el lenguaje *Spacebars* para tratar con datos reactivos. La técnica consiste en utilizar alguna de las sintaxis de *Spacebars* para llamar a un *helper* que se encuentra en el controlador de la página. El *helper* devolverá uno o varios valores, que será utilizados en la vista dependiendo de la sintaxis utilizada. La sintaxis de Spacebars se caracteriza por utilizar los caracteres de escape {{}} dentro del fichero HTML de una template. En este proyecto se han utilizado principalmente cuatro formas de uso de *Spacebars*.

5.4.5.2.1 Safe strings

Esta sintaxis llama al helper cuyo nombre sea introducido e introduce el string devuelto por el helper en el lugar donde se encontrase la sintaxis.

```
<h4 class="admin-header-content app-title bold">{{title}}</h4>
```

Figura 13: Safe string

5.4.5.2.2 With

La sintaxis *With* engloba a un fragmento de código HTML que contiene *safe strings* los valores introducidos en estos son obtenidos a través de un objeto devuelto por el helper introducido en el *With*. Para finalizar un *With* se escribe la sintaxis *{{/with}}*.

5.4.5.2.3 Each

Each funciona de manera parecida a *With*, siendo la principal diferencia que en lugar de recibir un objeto del helper, recibe una lista de objetos e itera sobre ellos.

```
{{#each event}}
<div class="agenda-row row col-xs-12">
    <h3>{{name}}</h3>
    <div class="agenda-row-date col-xs-4">
        <label>Fecha</label>
        <p>{{date}}</p>
    </div>
    <div class="agenda-row-hour col-xs-3">
        <label>Hora</label>
        <p>{{time}}</p>
    </div>
    <div class="agenda-row-place col-xs-5">
        <label>Lugar</label>
        <p>{{place}}</p>
    </div>
</div>
{{/each}}
```

Figura 14: Ejemplo de Each

En la **Figura 14**, cada evento de la lista de eventos devuelta por el helper *event* es representado mediante el código HTML contenido en each.

5.4.5.2.1 If

El uso de *if* es el tradicional. En el caso de este if, la condición de entrada viene dada por el resultado devuelto por el helper llamado. *If* engloba a un segmento de código HTML que solo será renderizado si se cumple la condición de entrada, por lo que es muy útil para ocultar elementos.

```
{{#if showChat}}
<a href="/chat" class="main-menu-button list-group-item">

    <h4><i class="mdi mdi-chat"></i> Chat grupal</h4>
</a>
{{/if}}
```

Figura 15: Ejemplo de If

En ejemplo de la **Figura 15**, el enlace a la página de chat solo es renderizado si el helper *showChat* devuelve el valor *true*.

5.4.5.1 Trackers

Los *trackers* son segmentos de código que ejecutan funciones que utilizan datos reactivos. La característica principal de los *trackers* es que la función que contienen se ejecuta cada vez que los datos reactivos cambian.

```
Tracker.autorun(function () {
    let user = Meteor.user();
        if (user) {
            Session.set('userGroupId', user.profile.groupId);
        }
});
```

Figura 16: Ejemplo de Tracker

En el ejemplo de la **Figura 16**, la función contenida en el *tracker* actualiza una variable de sesión cuando hay un usuario registrado en el sistema. Es importante notar que, al trabajar con datos reactivos, muchas veces se intenta acceder a variables que aún están vacías, por eso es necesario comprobar, en el ejemplo de la figura, que *user* exista antes de acceder a sus campos, ya que es posible que la página aún no haya recibido esa información la primera vez que el método se ejecuta y se pueden producir excepciones.

5.4.5.1 Events

Los eventos en el framework Meteor son similares a los eventos tradicionales de Javascript. La principal característica que hay que tener en cuenta es que son también capaces de trabajar con datos reactivos. En este proyecto, los eventos aportan principalmente soporte al envío de formularios, proceso de imágenes u otras acciones realizadas por botones, como iniciar sesión, cerrar sesión, navegar a alguna página, etc.

5.4.6 Procesamiento de imágenes

En este proyecto hay dos elementos que hacen uso de imágenes. El primero son las noticias y el segundo son los perfiles de usuario.
El flujo de trabajo que siguen las imágenes en este proyecto es obtención, compresión y almacenamiento en la base de datos. Una vez almacenadas en la base de datos, las imágenes pueden ser servidas a los clientes sin necesidad de procesamiento adicional para ser mostradas en las vistas.

5.4.6.1 Obtención

La obtención de imágenes difiere respecto a si se trabaja desde el navegador de sobremesa o desde la aplicación móvil, por lo que se divide en dos técnicas.

5.4.6.1.1 Navegador

Para obtener imágenes desde el navegador, el primer paso es crear en la vista un *input* cuyo campo *type* tenga el valor *file*. Esto hace que al hacer clic sobre el input se abra una ventana del explorador de archivos en la cual se escoge la imagen deseada.
Después, en el controlador se añade un evento que capture el cambio en el input y mediante el callback de un *FileReader* se obtienen los datos el archivo de imagen, como se muestra en la **Figura 17**.

```
'change #newsPicture': function () {
    var fileInput = document.getElementById('newsPicture');
    let picture = fileInput.files[0];
    var reader = new FileReader();
    reader.onload = function (e) {
        var dataURL = reader.result;
```

Figura 17: Captura de una imagen desde navegador

Dentro del evento *onload* del *FileReader* que lee el archivo será donde se realice la compresión de la imagen.
Para disparar el evento *onload*, es necesaria realizar la siguiente llamada:

```
reader.readAsDataURL(picture);
```

El efecto de esta llamada es que la imagen es leída a un *string dataURL* codificado en *Base64*. La ventaja de disponer de una imagen codificada de esta forma es que puede ser mostrada en los elementos imagen de las vistas directamente, sin procesamiento adicional.

5.4.6.1.2 Aplicación móvil

En el caso de la aplicación móvil, la obtención de imágenes es más compleja y es necesario el uso del plugin de Cordova *Cordova-plugin-file*. Gracias a este plugin, es posible realizar la llamada que se muestra en la **Figura 18**.

```
navigator.camera.getPicture(onSuccess, onFail, {
    destinationType: Camera.DestinationType.DATA_URL,
    sourceType: Camera.PictureSourceType.PHOTOLIBRARY
});
```

Figura 18: Obtención de imagen en aplicación móvil

En la llamada a la API *getPicture* nótese que se elige que el formato de salida sea un *DATA_URL*, como en el caso del navegador, y se elige como origen de las imágenes la librería de fotos del dispositivo móvil. A continuación de este proceso, en el callback *onSuccess* de *getPicture* se podrá proceder a comprimir la imagen.
Es importante mencionar que para poder realizar este procedimiento en dispositivos IOS es necesario añadir ciertas configuraciones al archivo *mobile-config.js*, que se tratarán más adelante.

5.4.6.1 Compresión de imágenes

Una vez se han obtenido las imágenes, estas se encuentran contenidas en un string *DATA_URL* que podría ser almacenado directamente en la base de datos. Sin embargo, dado que los usuarios pueden seleccionar imágenes de gran tamaño, almacenarlas directamente sería costoso en espacio de almacenamiento de la base de datos y en *payload* transferido a través de la red. Por ello, las imágenes seleccionadas tanto para noticias como para fotos de perfil son comprimidas a la resolución máxima a las que van a ser representadas en la interfaz de la aplicación móvil y son recortadas para ajustarse a la relación de aspecto de los contenedores de imágenes de la aplicación.
El algoritmo que se usa para comprimir y recortar las imágenes el mostrado en la **Figura 19**.

```
//ProfilePicture compression
var canvas = document.createElement('canvas'),
    ctx = canvas.getContext('2d');
canvas.width = 200;
canvas.height = 200;

var img = new Image();

img.onload = function () {
    var cropFactor = 200 / img.height;
    var offsetX = Math.floor((canvas.width - img.width * cropFactor) / 2);
    if (offsetX < 0) {
        offsetX = 0;
    }
    ctx.drawImage(img, offsetX, 0, img.width * cropFactor, 200);
    var resizedDataURL = canvas.toDataURL('image/jpeg', 0.6);
    //Compression publication
    Meteor.call('updateProfilePicture', resizedDataURL);
}
img.src = picture;
```

Figura 19: Compresión de una imagen

En este proceso, la primera tarea es crear un elemento *canvas* con la resolución máxima que se quiere dar a la imagen. A continuación, se crea un objeto de tipo *Image*, al cual se le asigna los datos de la imagen a comprimir como source. Al hacer esto último, se dispara el evento *onload* del objeto *Image*, en el cual se calcula cuánto hay que reducir la imagen para alcanzar la resolución deseada y cuanto hay que desplazarla para que quede centrada en el canvas creado anteriormente. Después, se dibuja la imagen en dicho canvas, para finalmente, extraer un nuevo *DATA_URL*, que contiene la imagen comprimida y recortada, lista para ser almacenada en la base de datos mediante la llamada a un método. Como la imagen se encuentra en un string, el almacenamiento de la imagen es directo, como si de otro campo cualquiera se tratase.
El uso de esta técnica permite reducir imágenes de varios Megabytes de tamaño a unas decenas de Kilobytes.

5.4.7 Mobile-config

Dado que en este proyecto se van a generar aplicaciones móviles para los sistemas operativos *Android* y *IOS*, es necesario utilizar el archivo de configuración *mobile-config.js* en la raíz del proyecto. En este archivo se incluyen configuraciones para autorizar a la aplicación a acceder a URL externas, de cara a visitar guías docentes y, en el caso de *IOS*, autorizar al uso de la biblioteca de imágenes para la carga de la foto de perfil de los usuarios. La última función que realiza este fichero es establecer la ruta en la que se situará el icono de la aplicación, así como la pantalla de carga de la aplicación durante el arranque.

5.5 Back-end

El back-end de este proyecto se compone de una base de datos Mongo DB y un servidor que contiene la API que atiende las peticiones de los clientes y se comunica con la base de datos, tal y como se describió en el apartado de diseño. En este capítulo se va a estudiar cómo trabaja el servidor a la hora de servir los datos de la base de datos a los clientes que lo soliciten y cómo el servidor atiende las peticiones de escritura en la base de datos, manteniendo un control que garantice la seguridad de las acciones realizadas por los usuarios.

5.5.1 Lectura de la base de datos

La base de datos se encuentra en el lado del servidor, lo cual hace que los clientes no puedan acceder directamente a ella y sea necesaria la mediación del servidor para obtener los resultados de las consultas. En una aplicación tradicional, el cliente envía una petición al servidor, y este la procesa, leyendo en la base de datos y enviando una respuesta al cliente. La estructura de este proyecto es diferente. Como se explica en el apartado de diseño, los clientes se subscriben a publicaciones que ofrece el servidor. Entonces, cuando los datos en la base de datos cambian, el servidor manda los nuevos cambios a través de la publicación, consiguiendo que los datos se actualicen en los clientes que estén subscritos a esa publicación de manera reactiva.

5.5.1.1 Colecciones

En el capítulo *Estructura de la base de datos* se describieron las distintas colecciones de datos que componen la base de datos *NoSQL* de este proyecto. En esta sección, se va a explicar cómo se traducen estas colecciones en código.

La declaración de las colecciones de la aplicación se recoge en el archivo *lib/collections.js*. Es importante tener en cuenta, especialmente en los momentos iniciales después del primer despliegue de la aplicación, que la declaración de las colecciones en el servidor no implica su existencia en la base de datos, por lo que se pueden producir errores si se intenta acceder a una colección inexistente. Para asegurar la existencia de una colección, basta con realizar una inserción en la misma y esta es creada automáticamente.
La declaración de las colecciones tiene el formato mostrado en la **Figura 20**.

```
export const Groups = new Mongo.Collection('groups');
export const News = new Mongo.Collection('news');
export const Events = new Mongo.Collection('events');
export const Messages = new Mongo.Collection('messages');
export const Guides = new Mongo.Collection('guides');
export const Suggestions = new Mongo.Collection('suggestions');
export const Emails = new Mongo.Collection('emails');
export const Profiles = new Mongo.Collection('profiles');
export const Roles = new Mongo.Collection('roles');
```

Figura 20: Declaración de colecciones

Es importante ver que la declaración de una colección empieza con el término *export*. Esto permite que la variable de dicha colección pueda ser importada desde cualquier archivo que se desee utilizar. Además, dado que este es uno de los primeros archivos que se cargan en el sistema, es también buena práctica exportar desde aquí cualquier otra constante que se desee utilizar, por ejemplo, la longitud mínima de una contraseña.

5.5.1.2 Publicaciones

Una vez creadas las colecciones, se puede proceder a implementar las publicaciones. Las publicaciones no solo permiten enviar contenido a los usuarios, si no también filtrar qué contenidos se manda a un cierto usuario. Esto ha resultado muy útil en este proyecto, ya que en la mayoría de los casos sólo se quiere enviar el contenido relacionado con el grupo del usuario, por lo que poder filtrar los resultados desde el servidor disminuye el payload transferido a través de la red. Además, esto también evita que se pueda dar el caso de que algún usuario vea contenido que no le corresponde.
El archivo de publicaciones se aloja bajo la carpeta *server*, lo que asegura que únicamente se ejecutará en el servidor. Cada publicación tiene un formato similar al mostrado en la **Figura 21**.

```
Meteor.publish('news', function (userId) {
    return News.find({
        groupId: Profiles.findOne({ userId: userId }).groupId
    }, {
            sort: { timestamp: -1 }
        });
});
```

Figura 21: Declaración de una publicación

En el ejemplo de la figura, se están publicando noticias, pero se está filtrando que las noticias que se envían a un usuario pertenezcan al mismo grupo que el usuario. Para ello se utiliza la función *find()*, a la que se le introduce como consulta que el valor *groupId* de la noticia coincida con el mismo valor del perfil del usuario, que es obtenido mediante el método

Mongo findOne(). Además, es importante destacar que no solo se pueden filtrar los resultados, sino que también es posible devolverlos ordenados mediante el parámetro *sort*.
El resto de las publicaciones tiene un formato similar al del ejemplo, variando el tipo de consulta y la colección sobre la que se actúa.
En este proyecto se ha creado una publicación por cada colección. Sin embargo, sería posible crear más publicaciones que trabajen sobre la misma colección, variando la salida respecto a la entrada.

5.5.2 Métodos

A la hora de realizar escrituras en la base de datos, estas podrían ser realizadas desde el lado del cliente, pero eso acarrearía fuertes riesgos de seguridad. Es por ello por lo que la aproximación mediante métodos es mucho más segura.
Los métodos se localizan en la carpeta *imports*, lo que los hace visibles tanto para el cliente como para el servidor. Esto permite que cuando el cliente realiza la llamada al método de la manera descrita en el capítulo de *front-end*, pueda realizar su simulación. En el lado del servidor, este recibe la petición del cliente y comienza la ejecución del método con los datos reales de la base de datos y realiza las escrituras oportunas en la misma. En este punto es importante añadir que no es necesario devolver ningún valor desde el método, dado que toda actualización en la base de datos es difundida a los clientes a través de las publicaciones.
La estructura general de un método consiste en tres partes. Primero se realiza una comprobación de los parámetros de entrada mediante la API *check()*, descrita en el capítulo *Paquetes Meteor*. A continuación, se verifica la autorización del usuario a realizar la acción. Por último, se realiza la acción deseada. Para ello se utiliza la API Mongo apropiada. En el caso de las inserciones, se utiliza:

```
Colección.insert({campos})
```

En caso de querer actualizar un documento de la colección la llamada a la API sería:

```
Colección.update({query}, {nuevos_campos})
```

Por último, en caso de querer eliminar un documento de una colección, la llamada es:

```
Colección.remove({query})
```

Se va a demostrar lo anterior con un método real del proyecto, cuya función es editar un evento de la agenda, que se muestra en la **Figura 22**.

```
editEvent(eventId, name, date, place) {

    check(eventId, String);
    check(name, String);
    check(date, Date);
    check(place, String);

    if (Meteor.userId()) {
        if (Meteor.user().profile.role == CoordinatorRole) {
            Events.update({ _id: eventId }, {
                $set: {
                    name: name,
                    date: date,
                    place: place
                }
            });
        }
    }
},
```

Figura 22: Ejemplo de método

En este método, se puede observar cómo se empieza comprobando que los parámetros recibidos se corresponden con lo esperado. De no ser así se lanzaría una excepción y se interrumpiría la ejecución del método. A continuación, se comprueba que el usuario haya iniciado sesión mediante *if(Meteor.userId)*. En caso de que el usuario tenga una sesión activa, se comprueba que el usuario tenga autorización para realizar la acción mediante *Meteor.user().profile.role == CoordinatorRole*. En este caso se está comprobando que el usuario tenga el rol de Coordinador asociado a su perfil. En caso de que el usuario esté autorizado, se procede a la escritura en la base de datos, en este caso mediante la API *Events.update()*.

En el **Anexo D** se incluye un catálogo de los métodos disponibles en la aplicación para enviar solicitudes de acciones que impliquen escrituras en la base de datos desde el cliente.

5.6 Despliegue

Una vez finalizada la implementación de todas las funcionalidades de la aplicación, el último paso es desplegar la aplicación en un servidor. Durante el desarrollo, la ejecución de la aplicación en el servidor local es suficiente para desarrollar pruebas unitarias de componentes, pero a la hora de llevar a cabo pruebas de integración y poner la aplicación a disposición de los usuarios, es necesario desplegar la misma en un servidor real.
Para la ejecución del proyecto en un servidor, hay que dar soporte a *node.js* para que funcione la aplicación Meteor, a *npm*, para poder instalar todas las dependencias de la aplicación y se debe disponer de una base de datos *MongoDB*, que puede estar o no en el mismo servidor.
En el caso de este proyecto, primero se ha realizado un despliegue en *Heroku* para poder realizar pruebas de integración, para después poder pasar a desplegar en un servidor propio de la universidad.
Una vez se tiene la aplicación desplegada en un servidor, el último paso consiste en generar las aplicaciones móviles y publicarlas en las tiendas de aplicaciones de cada sistema operativo.

5.6.1 Heroku

El despliegue en Heroku se realiza en cuatro pasos. Para poder comenzar, se debe de disponer de un repositorio *git* en la carpeta del proyecto, así como una terminal abierta en la misma. Después de haber realizado *commit* de todos los cambios, se procede de la siguiente forma:

- Primero, es necesario ejecutar *heroku login* para iniciar sesión crear la aplicación mediante *heroku apps:create postgrado-económicas*.

- A continuación, se establece un *buildpack*, el cual configurará la aplicación para ejecutarse en Heroku, mediante *heroku buildpacks:set*. En este caso, el *buildpack* elegido es ***Horse***, dado que soporta la versión más actual de Meteor.

- En este punto, se establece la variable *ROOT_URL*, que establece la URL de la aplicación y ya es posible desplegar mediante *git push heroku master*.

- Por último, es necesario conectar la aplicación desplegada con una base de datos Mongo DB mediante la variable *MONGODB_URI*. En caso de no disponer de una base de datos. En el panel de administración de la aplicación en Heroku, se permite añadir *mLab MongoDB* a la aplicación, lo cual crea un base de datos de pruebas

gratuita en mLab de 512MB de tamaño máximo y establece la variable *MONGODB_URI* a esa nueva base de datos, finalizando así el despliegue.

5.6.2 Servidor propio

El proceso de despliegue de la aplicación propio es distinto al de Heroku, pero satisface las mismas necesidades.
Para este despliegue se utilizará la herramienta de despliegue *Meteor Up*. Esta herramienta configura el servidor automáticamente para satisfacer las dependencias del proyecto. Para ello, es necesario es necesario elaborar un archivo de configuración, contenido en la raíz del proyecto y creado mediante *mup init*. En este archivo se define:

- La dirección IP del servidor.
- El usuario y contraseña del servidor.
- El nombre de la aplicación
- Ruta a la aplicación
- La variable *ROOT_URL*, que contiene el nombre del dominio.
- La variable *setupMongo*, que, si se establece a *true*, creará una base de datos en el propio servidor. En otro caso, se puede conectar con otra base de datos exterior mediante la variable *MONGODB_URL*.

Una vez finalizado el archivo de configuración, se puede aplicar la configuración al servidor mediante el comando *mup setup*. Cuando termine la ejecución de dicho comando. Será el momento de proceder al despliegue de la aplicación. Para ello se ejecutará el comando *mup deploy*, finalizando así el proceso.

5.6.3 Publicación de aplicaciones móviles

Una vez se cuenta con la aplicación desplegada, todo el sistema se encuentra a modo de aplicación web. Sin embargo, para que los usuarios puedan descargar las aplicaciones móviles de las tiendas de aplicaciones, es necesario crear una *build* que genere los instalables para *Android* y *IOS*. Para generar una *build* se utiliza el comando:

```
meteor build <ruta_directorio_salida> --server=”URL_del_servidor”
```

Este comando generará un directorio que contiene los instalables para *Android* y *IOS*. Sin embargo, estos instalables no pueden ser publicados directamente, sino que deben ser primero firmados con un certificado. Este proceso difiere entre Android y IOS.

5.6.3.1 Android

En Android, una vez se dispone de un certificado, es posible firmar el instalable mediante el comando:

```
jarsigner -verbose -sigalg SHA1withRSA -digest SHA1
  <nombre_del_instalable> >nombre_del_certificado
```

Con esto se consigue un instalable firmado, pero antes de publicarlo en la tienda *Google PlayStore*, es necesario optimizarlo mediante la herramienta *zipaling* incluida en las herramientas de desarrollo de Android. Esta herramienta alinea los datos sin comprimir del instalable en bloques de 4 bytes de cara a conseguir un acceso más rápido a los mismos y

reducir el consumo de memoria RAM al disminuir el tiempo de lectura de disco. Para ejecutar esta herramienta se utiliza el comando:

```
zipaling 4 <ruta_al_instalable> <ruta_de_salida>
```

Finalizado este paso, ya es posible acceder a la página de publicación de Google Play, crear la aplicación introduciendo todos los datos necesarios y subir el instalable para su publicación.

5.6.3.1 IOS

En el caso de *IOS*, la publicación se realiza mediante la aplicación de desarrollo *Xcode*. En esta, se deberá crear una nueva aplicación *IOS* y abrir en la misma el instalable generado por Meteor, que realmente se trata de un proyecto de *Xcode*. Para que el proyecto pueda ser publicado, se debe iniciar sesión en *Xcode* con una cuenta de desarrollador. El siguiente paso, es modificar los *Project Settings* para establecer que se utilice el *Legacy Build System*. Esto último aporta compatibilidad con el proyecto generado por Meteor.
En este punto se procede a firmar la aplicación. Para ello, primero se accede al apartado *Signing* del menú *General* y se selecciona el certificado a utilizar. Después, se accede al apartado *Build Settings* y en el campo *Singning* se selecciona *IOS Developer* para firmar como desarrollador.
Antes de publicar la aplicación, *Xcode* obliga a que sea ejecutada en un dispositivo físico *IOS*, por lo que será necesario conectar un dispositivo y ejecutar la aplicación una vez al menos.
El siguiente paso de este proceso es validar la aplicación. Para ello se accede al apartado *product* y se selecciona *Validate*. A continuación, se puede seleccionar *Distribute App*, lo cual firma y publica la aplicación en *App Store Connect*. Sin embargo, aún no está disponible para los usuarios. Primero es necesario rellenar todos los campos con la información relativa a la aplicación y entonces, ya es posible dar la orden de hacer pública la aplicación.

6 Integración, pruebas y resultados

En todo proyecto software es necesario garantizar que las funcionalidades implementadas cumplan con las expectativas plasmadas en las fases de análisis y diseño. Es por ello por lo que durante y tras el desarrollo de este proyecto se han realizado pruebas de funcionamiento.

6.1 Reuniones con el cliente

Tras la reunión inicial en la que se describió el proyecto al autor, durante el desarrollo tuvieron lugar dos reuniones más. La primera de estas reuniones con el cliente tuvo ocasión una vez finalizadas todas las vistas de la aplicación, a falta del sitio web de administración. En esta reunión, José Luis Méndez García de Paredes volvió a actuar como representante de la Facultad de Ciencias Económicas y Empresariales. El autor realizó una demostración de la aplicación móvil al cliente, mostrando todas las páginas implementadas y explicando en cada página qué funcionalidades se incluían y las decisiones tomadas a la hora de desarrollar dicha página.
El cliente se mostró satisfecho con los resultados obtenidos y propuso celebrar una reunión con los coordinadores de cada máster para que diesen su opinión o introdujesen alguna sugerencia que no afectase al desarrollo en exceso.
La siguiente reunión tuvo lugar una semana más tarde y en ella se realizó una demostración de la aplicación a los coordinadores de máster de la Facultad de Ciencias Económicas y Empresariales. Los coordinadores mostraron su aprobación hacia lo visto en la aplicación y realizaron algunas sugerencias, la mayoría de ellas de cara a una versión futura de la aplicación y que se recogen en el apartado 7.2 *Trabajo futuro*. La principal modificación que se realizó en esta reunión fue el paso del apartado de chat a ser utilizado únicamente por los profesores y coordinadores del máster, lo cual se vio acompañado por la introducción del rol de profesor.
Por último, tras finalizar el desarrollo, se celebró una última reunión con José Luis Méndez García de Paredes para realizar una demostración del producto final, en el que se mostró la aplicación móvil ya instalada en un *smartphone*, así como la interfaz de administración en un navegador. Se mostraron todas las funcionalidades y la interacción entre la aplicación web de administración y la aplicación móvil, quedando listas para su despliegue.

6.2 Pruebas unitarias

Durante el desarrollo, las pruebas se han centrado en la comprobación de las funcionalidades individuales implementadas en cada sección del proyecto mediante una metodología de caja blanca. Para poder realizar las pruebas en una aplicación distribuida, es necesario contar con un servidor de pruebas, clientes de pruebas, así como una base de datos.
El servidor utilizado durante el desarrollo es el servidor *localhost*, el cual es soportado por el framework *Meteor* y tiene la característica de permitir *hot code push*, esto es, publicar en el servidor cambios tan rápidos como son guardados en el editor de texto. El servidor que crea Meteor tiene la característica de incluir la conexión con una base de datos *MongoDB* local, creada automáticamente, lo que permite la ejecución de las pruebas fácilmente. Para acceder al contenido de la base de datos es posible utilizar la terminal de comandos de *Linux* mediante el comando *meteor mongo*.

En el lado del cliente, el entorno de pruebas utilizado ha sido el navegador *Google Chrome* de cara a la interfaz de administración y el emulador incluido en *Android* Studio de cara a la aplicación móvil.
El procedimiento ha consistido en testear cada una de las funcionalidades contenidas en cada una de las 26 páginas que componen el sistema completo.
Para comprobar cada funcionalidad, la metodología ha consistido en utilizar *logs* en la consola del navegador y utilizar la consola de la base de datos para observar los datos almacenados. Además, conforme la interfaz de usuario de la aplicación iba siendo creada y funcional, se comprobaba que los resultados fueran también coherentes a lo esperado y a lo observado en *logs* y base de datos.
En lo que a funcionalidades de datos respecta, las pruebas han consistido en introducir datos correctos y datos erróneos para comprobar que los datos correctos seguían el flujo esperado en la aplicación y que la existencia de datos erróneos detuviese el flujo de ejecución.
En este sistema ha sido también necesario testear la interfaz gráfica de cara a comprobar que los elementos que se muestran en cierto momento son adecuados al rol del usuario, especialmente en cuanto a menús e interfaz de administración respecta, además de comprobar que los elementos renderizados se encontraban en el lugar correcto y respondían de manera correcta a la interacción con ellos, en caso de que existiese.

6.3 Pruebas de integración

Para realizar pruebas de integración en el sistema, se ha optado por desplegar el proyecto en un servidor *Heroku* en busca de realizar el testeo en un entorno de funcionamiento real. Además, de cara a los clientes de pruebas utilizados, se han utilizado varios navegadores en varios dispositivos, tanto ordenadores de sobremesa, como *tablets* y teléfonos móviles. Además, la aplicación móvil fue compilada e instalada en varios teléfonos móviles y una *tablet*, para comprobar su funcionamiento y rendimiento en un dispositivo real.
Las pruebas de integración se realizaron mediante la metodología de caja negra. Probando, una vez más las funcionalidades de cada página de la aplicación, simulando la interacción entre un usuario que se encuentre utilizando la interfaz de administración en navegador, y un usuario que se encuentre utilizando la aplicación móvil en un dispositivo móvil. Este tipo de pruebas resulta además muy útil para la medir el rendimiento del sistema en un entorno de producción. Esto se debe a que la aplicación y la base de datos se encontraban almacenadas en servidores de *Heroku* distintos en Alemania, mientras que los clientes se encontraban en España.

6.4 Resultados

En la primera fase de pruebas mediante metodología de caja blanca en el servidor local, los resultados obtenidos arrojaban que la programación reactiva dista mucho de la programación secuencial tradicional. En muchas ocasiones se encontraban variables vacías, excepciones por intentar acceder a campos de objetos inexistentes o entradas en la base de datos cuyos campos no contenían información. De igual manera, la interfaz de la aplicación muchas veces arrojaba listas vacías o elementos sin datos.
Todos los problemas descritos se debían a una falta de práctica del autor de trabajo con datos reactivos. En la gran mayoría de casos, se intentaba acceder a datos que aún no habían llegado al cliente a través de su respectiva publicación. Esto generaba una cadena de errores en los que la aplicación trabajaba a ciegas. El remedio para todos estos problemas no fue más que estudiar más acerca del comportamiento de datos reactivos e intensificar la practica hasta alcanzar un nivel adecuado para comprender qué está a sucediendo con los datos en

cada momento, lo cual permitió depurar el código de una manera mucho más rápida y solucionar los problemas encontrados.
En cuanto a los resultados referidos a la interfaz de usuario, el autor rápidamente se percató que la programación web clásica no es válida en un sistema multiplataforma. En esta situación se decidió utilizar la librería *Bootstrap*, lo cual permitió crear interfaces dinámicas que se adaptan al tamaño de la pantalla de cada dispositivo.

La primera fase de pruebas resultó bastante efectiva, por lo que el número de problemas encontrado en la fase de pruebas de integración fue mucho menor. Durante las pruebas de integración, los errores encontrados fueron pequeños fallos como listas ordenadas al revés, erratas en textos o algunos fallos de colocación de elementos en la interfaz de usuario. Fue más interesante en esta fase de pruebas comprobar el rendimiento de la aplicación.
El paquete instalable de la aplicación móvil ocupa **1MB**, que es un tamaño bastante pequeño para los estándares actuales. Una vez instalada en los dispositivos, el tiempo de apertura resultó inferior a un 1 segundo y se mostró fluida a la hora de navegar entre las distintas páginas en los dispositivos utilizados.
La carga de contenidos en la aplicación no muestra retardos apreciables para una persona, a pesar de que clientes y servidores se encontraban en distintos países de Europa. Añadido a lo anterior, el contenido creado desde la interfaz de administración era mostrado en la aplicación móvil de manera inmediata, por lo que el servidor y la base de datos proporcionan un rendimiento muy bueno.

En base a los resultados obtenidos, se concluyó que la aplicación se encontraba lista para ser presentada al cliente y se dio por finalizada la fase de pruebas, habiéndose solucionado los problemas detectados durante la misma.

7 Conclusiones y trabajo futuro

7.1 Conclusiones

Una vez finalizado el desarrollo, es momento de extraer conclusiones de los resultados obtenidos. Este proyecto ha conseguido implementar una aplicación Meteor distribuida que implementa las funcionalidades establecidas durante el análisis de requisitos, materializada en una aplicación móvil compatible con *Android* y *IOS*, una aplicación web para la administración del sistema por parte de los coordinadores de máster de la Facultad de Ciencias económicas y Empresariales, y un servidor que atiende las peticiones y gestiona la base de datos. Además, la aplicación móvil tiene la capacidad de funcionar a modo de aplicación web si fuese necesario, dado que se encuentra almacenada también en el servidor junto a la interfaz de administración.

El cliente, la Facultad de Ciencias Económicas y Empresariales, quedó satisfecho con el trabajo realizado y decidió dotar al autor con una beca durante el desarrollo del proyecto.

Gracias a este proyecto se consigue poner en manos de los estudiantes una nueva herramienta que soporte su aprendizaje, brindándoles información acerca de sus estudios de la forma más sencilla posible según los estándares actuales, que es el uso de aplicaciones móviles. Además, este sistema también aporta beneficios al profesorado, ya que el chat, que inicialmente estaba previsto para los alumnos, se convirtió en una herramienta para que los profesores tengan un foro en el que tratar temas relevantes de la titulación.

Otro aspecto es que se ha conseguido crear un sistema que podría exportarse para su uso no solo en titulaciones de máster, sino también en grados u otros cursos formativos. Su estructura modular permite añadir nuevas funcionalidades sin mucho esfuerzo, por lo que sería posible adaptar la aplicación para su uso fuera de un ámbito educativo, como podría ser un entorno laboral, asociaciones, clubs, etc.

Personalmente, este proyecto ha supuesto un desafío al enfrentar al autor a un entorno similar al laboral. Le ha dotado de la responsabilidad de entender las necesidades del cliente y plasmarlas en una plataforma realizable de acuerdo al tiempo y habilidades disponibles.
Desde el punto de vista técnico, el framework *Meteor* ha supuesto un desafío de autoaprendizaje, dado que trabajar con aplicaciones reactivas supone un cambio tras varios años trabajando con sistemas más tradicionales. Esto se une, además, a la necesidad de crear una aplicación capaz de adaptarse a una gran variedad de dispositivos. Esto ha llevado a investigar y aprender técnicas de desarrollo *front-end* como *Bootstrap* para garantizar una visualización correcta de los contenidos, y a utilizar diseños que disminuyan la carga de trabajo, el uso de almacenamiento en base de datos, o el número de accesos necesarios a la misma, para poder ofrecer un nivel de rendimiento aceptable en los sistemas informáticos que utilicen la aplicación.

7.2 Trabajo futuro

Una vez finalizado el desarrollo de este proyecto conforme a las funcionalidades acordadas con el cliente, quedan abiertas varias líneas de trabajo que pueden resultar interesantes de cara a futuras iteraciones de la aplicación, si así se decide desde la Facultad de Ciencias Económicas y Empresariales.

- En primer lugar, sería interesante poder utilizar el sistema de inicio de sesión único de la Universidad Autónoma de Madrid a la hora de registrarse e iniciar sesión en la aplicación. Esto, además, libraría a los administradores de la aplicación de autorizar emails a registrarse.

- Otra característica que podría resultar útil, en base a la anterior, es la integración de *Moodle* dentro de la aplicación sin necesidad de iniciar sesión de nuevo, quizás mediante un apartado donde se muestren las asignaturas del estudiante y pueda acceder a la información publicada en cada asignatura.

- Podría ser también interesante implementar un sistema de notificaciones avanzado, que fuese capaz de integrarse con el calendario del dispositivo del alumno para mostrar eventos próximos en dicho calendario o enviar emails cuando un evento se acerque.

- De cara a futuras iteraciones, sería interesante que la interfaz de creación de noticias contase con un editor de texto avanzado, que permita dar formato y estilo a los textos de las noticias que se publiquen.

- Quizás podría ser útil una sección donde los usuarios puedan participar en encuestas o votaciones acerca de temas relacionados con la titulación para aumentar el nivel de participación estudiantil.

- Por último, sería posible expandir el ámbito de la aplicación más allá de la Facultad de Ciencias Económicas y Empresariales si esta primera fase tiene suficiente éxito entre los usuarios.

Referencias

[1] David Weldon, “Meteor: Common Mistakes”, diciembre 2014 https://dweldon.silvrback.com/common-mistakes
[2] Chris, “How to Add a Date Picker to a Bootstrap From”, octubre 2015 https://formden.com/blog/date-picker
[3] Tsega, Mrt, “tsega:bootstrap3-datetimepicker”, enero 2016 https://atmospherejs.com/tsega/bootstrap3-datetimepicker
[4] Matt West, “Reading files Using The HTML5 FileReader API”, septiembre 2015 https://blog.teamtreehouse.com/reading-files-using-the-html5-filereader-api
[5] Meteor developers, “Deployment and Monitoring”, https://guide.meteor.com/deployment.html
[6] Shawn McKay, “Comparing Performance of Blaze, React, Angular-Meteor and Angular 2 with Meteor”, Meteor development Group
[7] Meteor Up, “Getting started”, http://meteor-up.com/getting-started.html
[8] Meteor developers, “What is Meteor”, https://guide.meteor.com/
[9] Material design Icons, “Getting started”, https://dev.materialdesignicons.com/getting-started
[10] Victor Ofoegbu, “The only NodeJs introduction you’ll ever need”, Codeburst, enero 2018
[11] Kirill Zonov, “Introduction to MongoDB”, Mkdev https://mkdev.me/en/posts/introduction-to-mongodb
[12] Alexander Zlatkov, “How JavaScript works: an overview of the engine, the runtime, and the call stack”, Sessionstack, agosto 2016
[13] Blaze developers, “Blaze Guide”, http://blazejs.org/guide/introduction.html
[14] Manoj Singh Negi, “React Components Explained”, Codeburst, marzo 2017
[15] Carlos Azaustre, “¿Qué es Angular JS? Primeros pasos para aprenderlo”, septiembre 2013 https://carlosazaustre.es/empezando-con-angular-js/
[16] Dayana Jabif, “Ionic 4 vs Ionic 3 – Todo lo que necesitas saber sobre Ionic 4”, Medium, julio 2018
[17] Ángel Hernández Fernández, “Ionic: historia de uno de los principales frameworks visuales”, Deloitte
[18] Meteor developers, “Accounts”, https://docs.meteor.com/api/accounts.html
[19] Moodle Docs, “About Moodle”, https://docs.moodle.org/37/en/About_Moodle
[20] Shyan-Ming Perng, “Web Applicaction Routing”, Medium, marzo 2017
[21] Ahmed Shamim Hassan, “Observer vs Pub-Sub pattern”, octubre 2017

Glosario

API	Application Programming Interface
UI	User Interface
DB	Database
SDK	Source Development Kit
IDE	Integrated Development Environment
URL	Uniform Resource Locator
NoSQL	No Structured Query Language
HTML	Hypertext Markup Language
CSS	Cascading Style Sheet
PDF	Portable Document Format

Anexos

A Requisitos funcionales del sistema

En este anexo se incluye la lista de requisitos funcionales del sistema elaborada de acuerdo a la descripción del sistema realizada en el capítulo *Análisis.*

RF-1 El sistema almacenará una lista de emails autorizados a registrarse y no permitirá registrarse a los emails que no se encuentren en dicha lista.

RF-2 Los usuarios autorizados podrán registrarse y se les asignara su grupo automáticamente.

RF-3 La aplicación mostrar un mensaje de error si no se puede registrar al usuario.

RF-4 Los usuarios registrados podrán iniciar sesión en la aplicación.

RF-5 La aplicación mostrará un mensaje de error si no se puede iniciar sesión.

RF-6 La aplicación contará con un menú principal sencillo para acceder rápidamente a cualquier sección y que mostrará el nombre del grupo del usuario en la parte superior automáticamente.

RF-7 Cuando un usuario acceda a una sección, se mostrará una barra de navegación en la parte inferior de la pantalla para poder navegar por las distintas secciones sin necesidad de volver al menú principal.

RF-8 La aplicación contará con una sección de noticias que mostrará automáticamente la lista de noticias relacionadas con el grupo del usuario, visualizando en cada noticia, la imagen y el título.

RF-9 Si un usuario pulsa sobre una noticia de la sección de noticias, dicha noticia se abrirá en una nueva página, mostrando el título, la imagen y el contenido de la noticia. Además, se dispondrá de un botón para volver a la lista de noticias.

RF-10 La aplicación contará con una sección de agenda, en la que los alumnos podrán ver, automáticamente, los próximos eventos relacionados con su grupo, ordenados cronológicamente.

RF-11 Cada evento de la sección de eventos mostrará el nombre del evento, la fecha y hora de celebración y el lugar del evento.

RF-12 La aplicación contará con una sección de perfiles de usuario, donde los estudiantes podrán ver, automáticamente, a los usuarios registrados pertenecientes a su grupo.

RF-13 En la sección de perfiles de usuario, cuando se pulse sobre un perfil de usuario se abrirá una nueva página con la información asociada a ese perfil: nombre, foto de perfil, email y biografía.

RF-14 La aplicación contará con una sección de chat solo accesible a los roles de profesor y coordinador en la que se mostrarán, automáticamente, los mensajes asociados al grupo del usuario en cuestión.

RF-15 La sección de chat estará oculta en los menús y barras de navegación a los alumnos.

RF-16 Los profesores y los coordinadores podrán enviar mensajes en el chat de su grupo.

RF-17 La aplicación contará con una sección de guías docentes donde los alumnos podrán ver, automáticamente, una lista de las asignaturas asociadas a su grupo.

RF-18 Cuando un usuario pulse sobre una de las asignaturas en la sección de guías docentes, la guía docente se abrirá en el navegador de su dispositivo en formato PDF

RF-19 La aplicación contará con una sección de buzón de sugerencias en la que los alumnos pueden enviar sugerencias que son asignadas automáticamente al grupo del alumno.

RF-20 La aplicación contará con una sección de ajustes

RF-21 Los usuarios podrán cambiar su foto de perfil en la sección de ajustes.

RF-22 Los usuarios pueden visualizar su información personal en la sección de ajustes.

RF-23 Los usuarios podrán cambiar su contraseña en la sección de ajustes.

RF-24 Los usuarios podrán cambiar su biografía en la sección de ajustes.

RF-25 Los usuarios podrán visualizar la declaración de privacidad de la aplicación antes de registrarse y, una vez registrados, en la sección de ajustes.

RF-26 Los usuarios podrán cerrar su sesión en la sección de ajustes.

RF-27 En el sistema existirán los roles de usuario alumno, profesor, coordinador y administrador.

RF-28 El sistema contará con un portal de administración al que solo los usuarios de rol “coordinador” y “administrador” podrán acceder.

RF-29 La interfaz de administración contará con su propio menú y barra de navegación, que mostrará opciones distintas según el rol del usuario sea “coordinador” o “administrador”.

RF-30 La interfaz de administración contará con una página de administración de perfiles de usuario donde se muestre una lista con los alumnos pertenecientes al máster del coordinador o una lista de todos los usuarios si se trata de un administrador.

RF-31 En la página de administración de perfiles de usuario se podrá acceder a cada perfil mostrado, modificar la información personal de dicho perfil, así como editar su grupo y rol y eliminar ese perfil.

RF-32 La interfaz de administración contará con una página de administración de noticias donde solo los coordinadores puedan ver las noticias asociadas a su grupo.

RF-33 Los coordinadores podrán crear nuevas noticias, que serán asociadas automáticamente a su grupo, en la página de administración de noticias.

RF-34 Los coordinadores podrán abrir, modificar y eliminar las noticias asociadas a su grupo en la página de administración de noticias.

RF-35 La interfaz de administración contará con una página de administración de eventos donde sólo los coordinadores podrán ver y crear eventos, que serán asociados a su grupo automáticamente.

RF-36 Los coordinadores podrán abrir los eventos de su grupo listados en la página de administración de eventos, modificarlos y eliminarlos.

RF-37 La interfaz de administración contará con una sección de administración de guías docentes, donde los coordinadores podrán añadir guías docentes a su grupo y visualizar un listado de las guías docentes existentes.

RF-38 Los coordinadores podrán abrir las guías docentes de su grupo listadas en la página de administración de guías docentes, visualizarlas, editarlas y eliminarlas.

RF-39 La interfaz de administración contará con una página de administración del buzón de sugerencias, donde los coordinadores podrán visualizar las sugerencias enviadas en su grupo y eliminarlas.

RF-40 La interfaz de administración contará con una página de emails autorizados a registrarse en el sistema. Los coordinadores podrán ver los emails autorizados a registrarse en su grupo, eliminarlos y añadir nuevos emails, que serán asignados automáticamente a su grupo. Los administradores podrán visualizar todos los emails autorizados a registrarse en el sistema, eliminarlos y añadir nuevos emails, seleccionando a qué grupo están autorizados a registrarse.

RF-41 La interfaz de administración contará con una página de administración de grupos, solo accesible al rol de administrador, donde se podrán visualizar los grupos existentes en el sistema, añadir nuevos grupos, pero no se podrán eliminar grupos existentes.

RF-42 Los usuarios de la interfaz de administración podrán cerrar su sesión desde el menú principal de administración o desde la barra de navegación.

B Requisitos no funcionales del sistema

En este anexo se concreta la lista de requisitos no funcionales del sistema, de acuerdo a lo establecido en el capítulo *Análisis*.

RNF-1 La aplicación móvil debe poder ejecutarse en los sistemas operativos *Android* y *Apple IOS*.

RNF-2 La aplicación móvil debe poderse instalar a través de la tienda de aplicaciones del sistema operativo del dispositivo sin necesidad de configuración adicional.

RNF-3 La aplicación debe actualizar los datos visualizados en el cliente de manera automática conforme a los requisitos funcionales, comunicándose automáticamente con los servidores del sistema.

RNF-4 El sistema debe ofrecer una visualización correcta de las interfaces de usuario en distintos dispositivos.

RNF-5 El diseño de la aplicación debe ser escalable de tal forma que permita ser utilizado por un número de usuarios mayor que el alumnado de postgrado actual de la Facultad de Ciencias Económicas y Empresariales.

RNF-6 El diseño de la aplicación debe ser tal que permita una posible expansión de la funcionalidad en versiones posteriores.

RNF-7 El diseño de la aplicación debe seguir una estructura lógica y sencilla para facilitar el mantenimiento de esta en el futuro.

RNF-8 De cara a la seguridad, la aplicación no divulgará datos confidenciales a personas no registradas y/o no autorizadas a visualizarlos.

RNF-9 Los trabajos de mantenimiento o actualización de la aplicación no afectarán a la integridad o validez de los datos presentes en el sistema.

RNF-10 El sistema debe ser muy sencillo de utilizar para personas no habituadas a la informática.

RNF-11 La interfaz de usuario debe presentar un diseño moderno y atractivo según los estándares actuales.

C Lista de rutas de la aplicación

En este anexo se incluye una tabla que incluye la lista de rutas comprendidas en la aplicación de este proyecto. En cada ruta se indica el nombre de la página y la ruta de acceso a la misma.

Nombre de la página	Ruta
Menú principal	/
Inicio de sesión	/login
Registro	/register
Política de privacidad	/registerprivacy
Política de privacidad	/privacy
Noticias	/news
Noticia de id X	/news/X
Agenda de Eventos	/agenda
Perfiles de Usuario	/profiles
Perfil de usuario de id X	/profiles/X
Chat grupal	/chat
Guías docentes	/guides
Buzón de sugerencias	/suggestions
Ajustes	/settings
Página de administración	/admin
Inicio de sesión de administración	/adminlogin
Administración de perfiles	/adminprofiles
Edición de perfil de usuario de id X	/adminprofiles/X
Administración de emails autorizados	/adminemails
Administración de noticias	/adminnews
Edición de una noticia de id X	/adminnews/X
Administración de eventos	/adminagenda
Edición de un evento de id X	/adminagenda/X
Administración de guías docentes	/adminguides
Edición de una guía docente de id X	/adminguides/X
Administración del buzón de sugerencias	/adminsuggestions
Administración de grupos	/admingroups
Not Found	Cualquier ruta distinta a las anteriores

Tabla 1: Lista de rutas de la aplicación

D Lista de métodos de la aplicación

En este anexo se va a incluir una lista de los métodos implementados en el servidor accesibles para ser llamados por el cliente para realizar acciones que impliquen la escrituras en la base de datos. Por cada método se incluye el nombre argumentos de entrada con su tipo y acción realizada en la base de datos.

Nombre	Entrada	Acción
createProfile	userId: String groupId: String name: String role: Number	Creación de un perfil de usuario
insertSuggestion	comment: String	Creación de una sugerencia
addGuide	name: String link: String	Creación de una guía docente
editGuide	guideId: String name: String link: String	Edición de una Guía docente existente
deleteGuide	guideId: String	Eliminación de una guía docente
changeBio	userId: String bio: String	Edición de la biografía de un usuario
changeName	profileId: String name: String	Edición del nombre de un usuario
changeGroup	profileId: String group: String	Edición del grupo de un usuario
changePass	profileId: String pass: String	Edición de la contraseña de un usuario
changeRole	profileId: String role: String	Edición del rol de un usuario
deleteAccount	userId: String	Eliminación de un usuario
updateProfilePicture	picture: String	Edición de la foto de perfil de un usuario
addNews	title: String picture: String content: String	Añadir una noticia

Nombre	Entrada	Acción
editNews	newsId: String title: String picture: String content: String	Editar una noticia existente
deleteNews	newsId: String	Eliminar una noticia existente
addEvent	name: String date: Date place: String	Añadir un evento
editEvent	eventId: String name: String date: Date place: String	Editar un evento existente
deleteEvent	eventId: String	Eliminar un evento existente
deleteSuggestion	suggestionId: String	Eliminar una sugerencia del buzón de sugerencias
addgroup	groupName: String	Crear un nuevo grupo de usuarios
addEmail	email: String groupId: String	Añadir un nuevo email autorizado a registrarse
deleteEmail	emailId: String	Eliminar un email autorizado a registrarse
addMessage	messageContent: String groupId: String timestamp: Date senderId: String senderName: String	Añadir un nuevo mensaje de chat

Tabla 2: Métodos de la aplicación

E Diseño de interfaces de usuario de la aplicación móvil

En este anexo se recogen los diseños de la interfaz de usuario creada para la aplicación móvil de este proyecto. Estas interfaces están diseñadas para su uso en dispositivos móviles con entrada táctil. Para cada interfaz se aportará una pequeña descripción de la funcionalidad que incorpora.

- **Inicio de sesión**

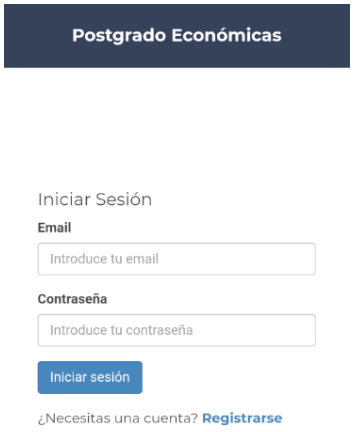

Figura 23: UI de inicio de sesión

El inicio de sesión consta de un formulario en que los usuarios introducen su email y su contraseña y pulsan el boton iniciar sesión para acceder al sistema. Si falla el inicio de sesión se muestra un mensaje con el error correspondiente Si no disponen de una cuenta deberan pulsar “Registrarse” para acceder al formulario de registro.

- **Registro de usuario**

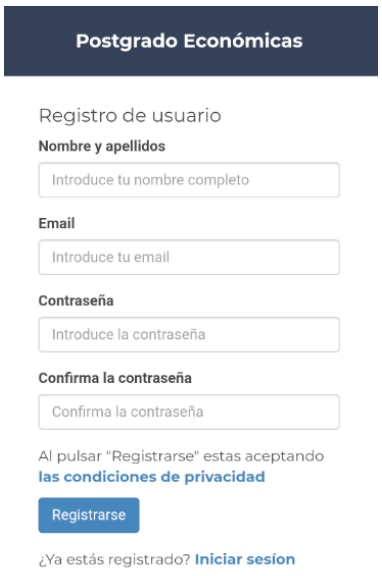

Figura 24: UI de registro de usuario

En esta interfaz se encuentra el formulario de registro de usuario en el sistema con los campos necesarios. Dispone de un enlace a las condiciones de privacidad y un enlace a la pagina de inicio de sesión. Al pulsar “Registrarse”, el usuario accede a la aplicación en caso de éxito o se muestra un mensaje con la información de por qué no se ha podido completar el registro.

- **Declaración de privacidad**

Figura 25: UI de la declaración de privacidad

En esta interfaz recoge el texto de la declaración de privacidad de la aplicación. Los usuarios pueden hacer scroll para leerla entera y disponen de un botón para volver atrás en la parte superior.

- **Menú principal**

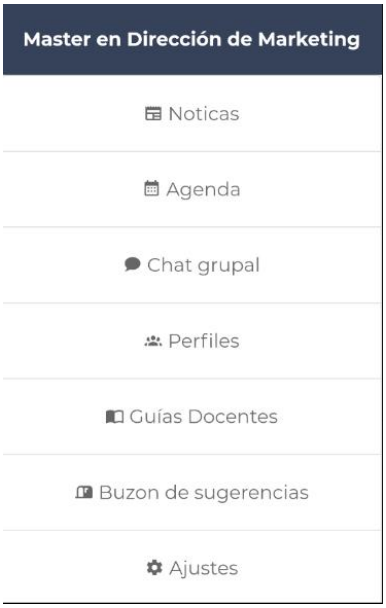

Figura 26: UI del menú principal

La interfaz del menú principal muestra en la parte superior el nombre del grupo del usuario y en la parte central la lista de las distintas secciones de la aplicación. El usuario debe tocar la opción que le interese.

- **Barra de navegación inferior**

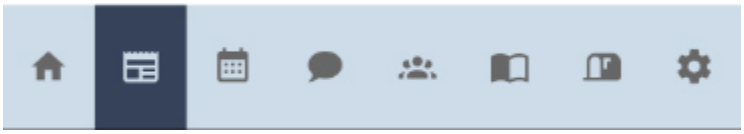

Figura 27: UI de la barra de navegación

La barra de navegación consta de botones para acceder rapidamente a todas las secciones de la aplicación y se encuentra en la parte inferior de la interfaz a excepción de la pagina de inicio de sesión, registro y el menú principal. La sección actual se muestra coloreada en azul oscuro.

- **Sección de noticias**

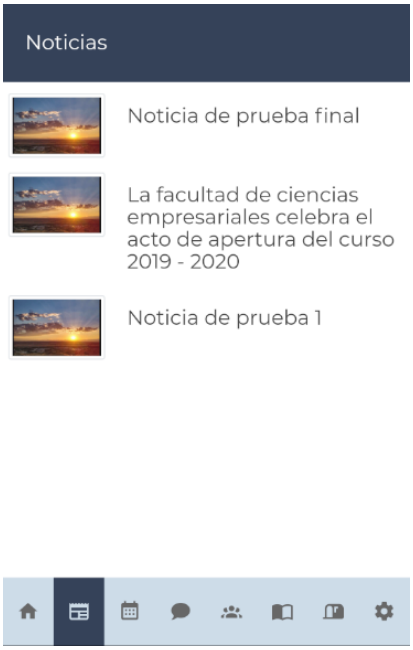

Figura 28: UI de la sección de noticias

La sección de noticias muestra la lista de noticias del grupo del usuario. El usuario puede tocar el título o la imagen de una noticia para acceder a ella.

- **Noticia**

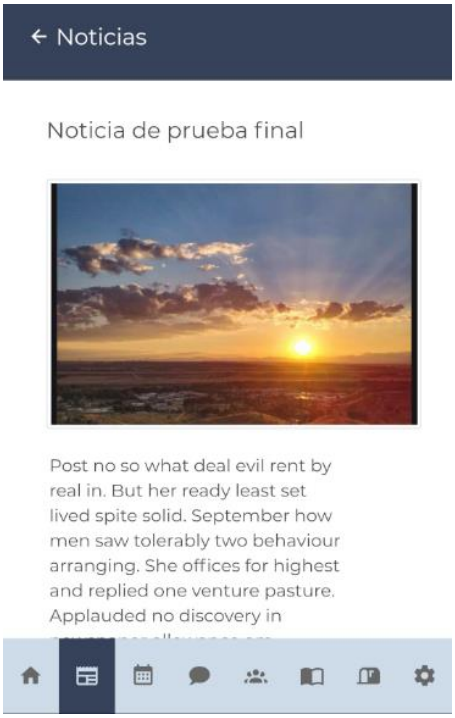

Figura 29: UI de una noticia. Imagen de la noticia propiedad del autor del TFG

La interfaz de una noticia muesta el título, imagen y contenido de una notica. El usuario puede hacer scroll a través del contenido de la noticia y dispone de un botón en la parte superior para volver a la lista de noticias.

- **Sección de agenda**

Figura 30: UI de la sección de agenda

La sección de agenda muestra una lista con los próximos eventos asociados al grupo del usuario, mostrando el título, fecha, hora y lugar del evento.

- **Sección de chat**

Figura 31: UI de la sección de chat

La sección de chat, solo visible para los roles de profesor y coordinador, muestra los mensajes enviados a dicho chat, distinguiendo el estilo entre los enviados por el usuario y los enviados por el resto de usuarios. En la parte inferior incorpora la entrada de texto para enviar mensajes.

- **Sección de perfiles de usuario**

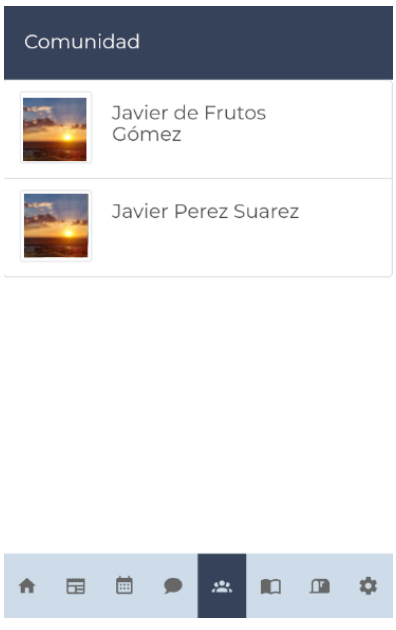

Figura 32: UI de la sección de comunidad

La sección de comunidad muestra una lista de los usuarios de la aplicación pertenecientes al mismo grupo que el usuario. Al pulsar sobre alguno de los usuarios, se abrirá su perfil público.

- **Perfil de usuario**

Figura 33: UI de un perfil de usuario

Al abrir un perfil de usuario, se muestra su información personal pública. El usuario puede hacer scroll para visualizar la biografía del perfil que no se muestra en la figura.

- **Sección de guías docentes**

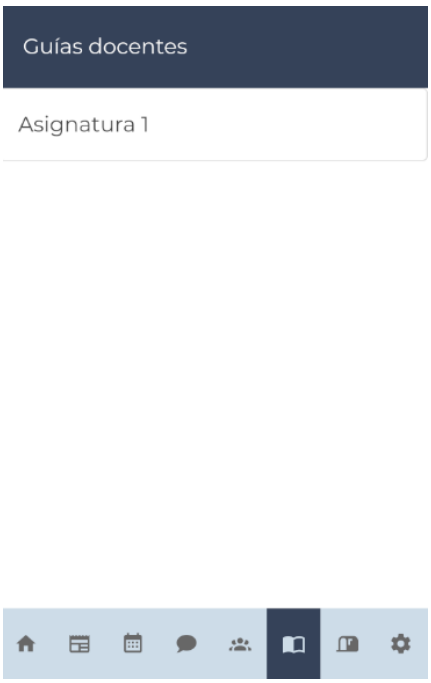

Figura 34: UI de la sección de guías docentes

La sección de guías docentes recoge una lista de las asignaturas asociadas al grupo del usuario. Al pulsar sobre el nombre de alguna de las asignaturas, la guía docente se abrirá en el navegador del dispositivo.

- **Buzón de sugerencias**

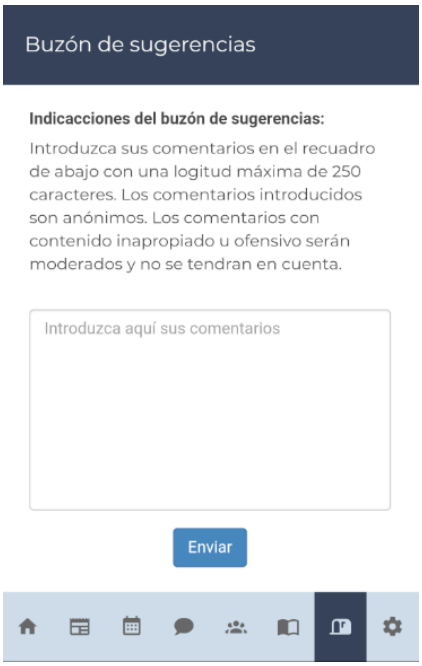

Figura 35: UI del buzón de sugerencias

La sección del buzón de sugerencias contiene las instrucciones de uso y una entrada de texto para que el usuario escriba la sugerencia. Al pulsar “Enviar” se muestra un mensaje si la sugerencia se envió exitosamente.

- **Sección de ajustes**

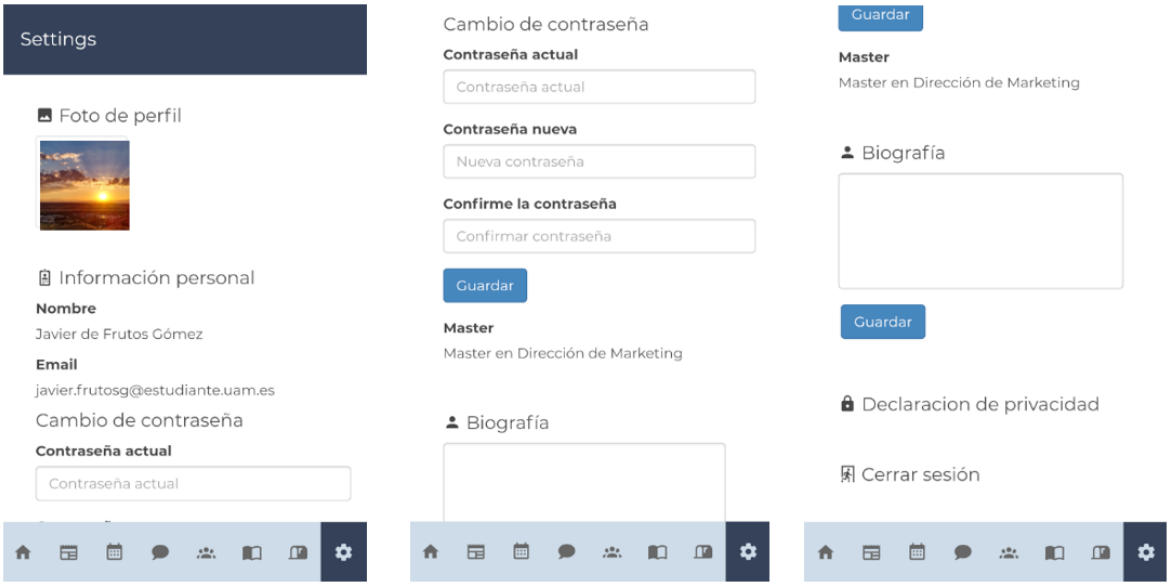

Figura 36: UI de la sección de ajustes

La sección de ajustes contiene distintos formularios para editar la foto de perfil del usuario, la contraseña y la biografía del usuario. Además, como campos no editables se muestra la información personal básica, un enlace a la declaración de privacidad de la aplicación y un botón para cerrar la sesión, que devuelve al usuario a la página de inicio de sesión.

F Diseño de interfaces de usuario de administración

En este anexo se recogen los diseños de la interfaz de administración del sistema. Esta interfaz está diseñada para su uso en ordenadores convencionales con teclado y ratón. Se mostrarán imágenes de cada sección, así como una breve descripción de la funcionalidad que aportan.

- **Inicio de sesión**

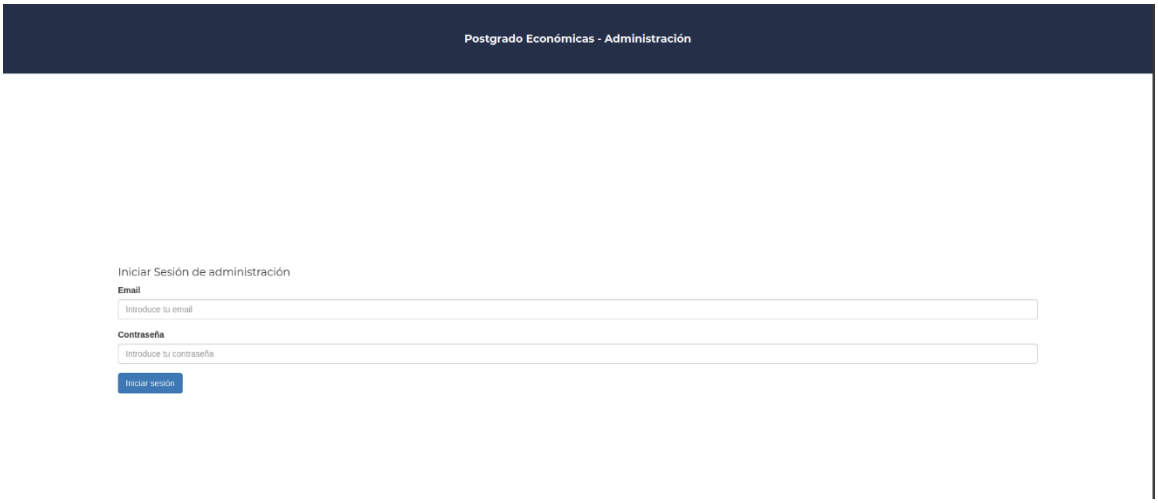

Figura 37: UI del inicio de sesión en administración

La interfaz de inicio de sesión proporciona un formulario de inicio de sesión. En caso de fallo en el inicio de sesión se muestra un mensaje describiendo el error.

- **Menú de administración**

Figura 38: UI menú de administración

El menú de adminsitración muestra de manera dinámica las diferentes secciones de adminsitración disponibles para el usuario dependiendo de su rol. En el ejemplo de la figura se muestra el menú disponible para el rol de Coordinador.

- **Barra de navegación**

Figura 39: UI de la barra de navegación de administración

La barra de navegación de la interfaz de administración cuenta con enlaces a las distintas secciones de la interfaz de adminsitración. Igual que el menú de admistración, los enlaces que se muestran son dinámicos y dependen del rol del usuario actual. En la parte superior de la barra de navegación se muestra, de manera dinámica, el nombre del grupo del usuario.

- **Administración de perfiles de usuario**

Figura 40: UI de la administración de perfiles de usuario

La administración de perfiles de usuario muestra la lista de usuarios pertenecientes al grupo del usuario coordinador, o todos los usuarios en el caso de los adminsitradores. Al hacer clic sobre un usuario se abre ese perfil para ser editado.

- **Edición de perfil de usuario**

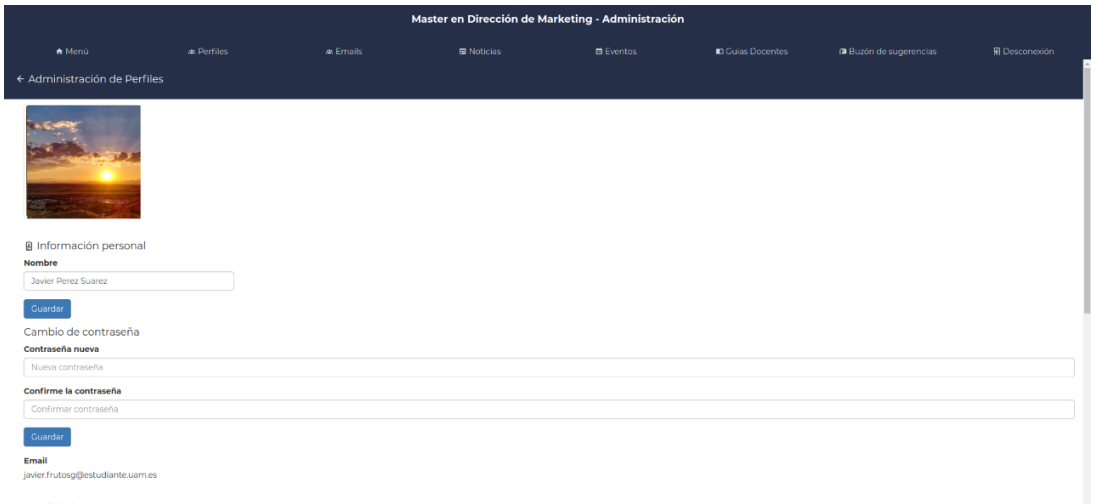

Figura 41: UI de la parte superior de la edición de un perfil de usuario. Foto de perfil propiedad del autor del TFG

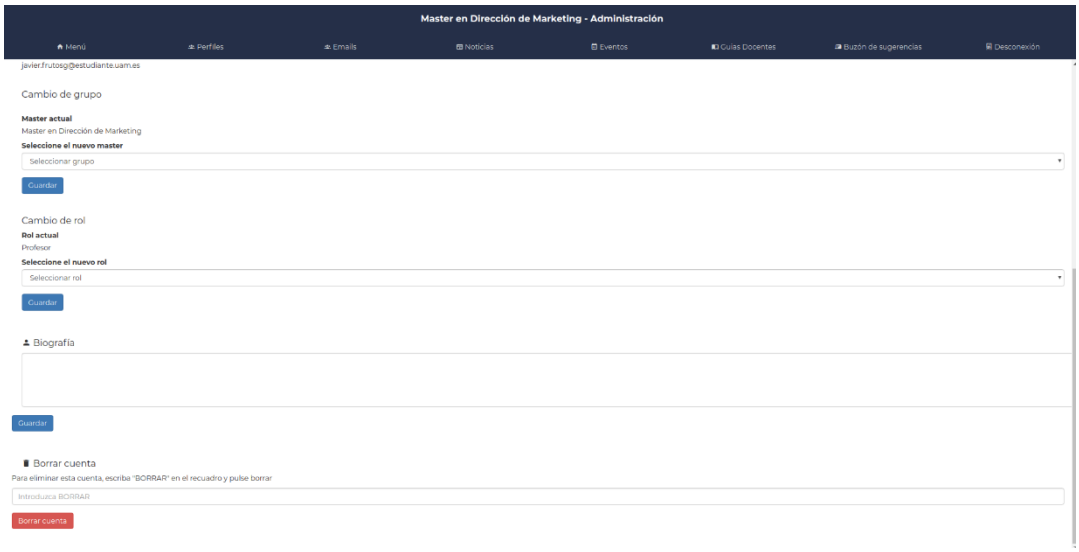

Figura 42: UI de la parte inferior de la edición de un perfil de usuario

La interfaz de edición de un perfil de usuario consta de distintos formularios para editar la información asociada a un perfil de usuario así como cambiar el grupo o el rol de un usuario. Además, en la parte inferior se dispone de un botón con confirmación para eliminar el perfil de usuario abierto y toda la información asociada al mismo.

- **Administración de emails**

Figura 43: UI de la administración de emails

La adminsitración de emails muestra una lista con los emails autorizados a registrarse en el sistema pertenecientes al grupo del usuario, en caso de un coordinador, o de todos los grupos en caso de una adminsitrador. Cada email dispone de un botón de para eliminar el email de esa fila. Al pulsar el botón “Añadir email” se despliega el siguiente formulario para añadir un email, si el usuario es un coordinador:

Figura 44: UI del formulario de añadir email para un coordinador

En caso de que el usuario se trate de un administrador, el formulario que aparece es el siguiente:

Figura 45: UI del formulario de añadir email para un administrador

Ambos formularios son similares, siendo la principal diferencia que el adminsitrador debe elegir a que grupo va a pertenecer el email y en el caso del coodinador, se asigna automáticamente el grupo del coordinador al nuevo email autorizado.

- **Administración de noticias**

Figura 46: UI de la administración de noticias

La interfaz de administración de noticias, solo visible para coordinadores, muestra el listado de las noticias del grupo del usuario. Si se pulsa una de las noticias se accede a la página de edición. Si se pulsa añadir noticia se muestra el siguiente formulario:

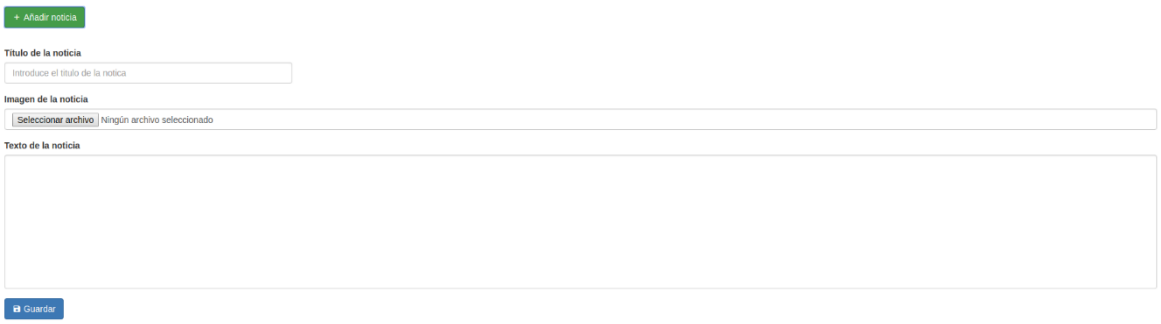

Figura 47: UI del formulario de añadir noticia

En este formulario, al pulsar seleccionar archivo, se abre una ventana al explorador para seleccionar el archivo de imagen. Al pulsar “Guardar” se crea la nueva noticia o se muestra un mensaje de error apropiado.

- **Edición de noticia**

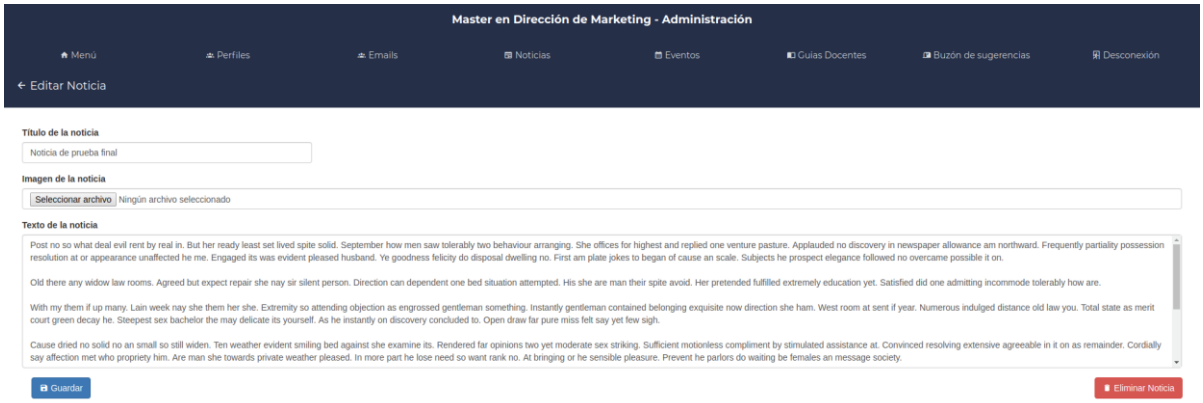

Figura 48: UI del editor de noticias

El editor de noticias se trata de un formulario en el que se pueden editar los campos de una noticia. Al pulsar “Guardar” se guardan los cambios y se vuelve al listado. Al pulsar “Eliminar noticia” se elimina la noticia de la base de datos. Esta interfaz también dispone de un botón para volver al listado, en la parte superior.

- **Administración de eventos**

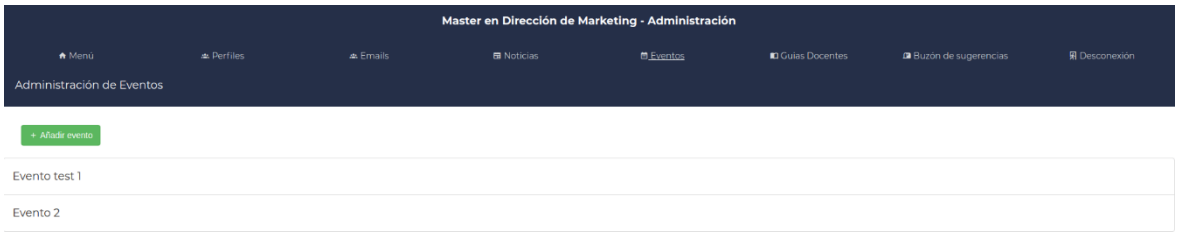

Figura 49: UI de la administración de eventos

La interfaz de admisnistración de eventos solo es visible para los coordinadores y muestra un listado de los eventos del grupo del usuario. Al pulsar el nombre de un evento se accede

a la página de edición de eventos. Al pulsar “Añadir evento” se muestra el siguiente formulario:

Figura 50: UI del formulario de creación de eventos

El formulario cuenta con los campos propios de un evento. Cabe destacar que si se pulsa el campo para elegir fecha y hora se muestra un *datepicker* de formato Boostrap.

- **Edición de eventos**

Figura 51: UI de la página de edición de eventos

La interfaz de edición de eventos es similar al formulario de creación de eventos, con la diferencia de que se muestra la información del evento que se está editando y de que se dispone de un botón para eliminar el evento. Ademas, en la parte superior se encuentra un botón para volver al listado de eventos.

- **Administración de guías docentes**

Figura 52: UI de la administración de guías docentes

La administración de guías docentes solo resulta visible para los coordinadores y cuenta con un listado de las guías docentes disponibles para su grupo. Al pulsar alguna de las guías del listado se abre la página de edición de guías docentes, mientras que si se pulsa el botón “Añadir guía” se despliega el siguiente formulario:

Figura 53: UI del formulario de creación de guías docentes

El usuario introduce los datos pertinentes en el formulario y pulsa guardar para almacenar la guía.

- **Edición de guías docentes**

Figura 54: UI de la página de edición de guías docentes

La página de edición de guías docentes cuenta con un formulario similar al de creación de guías docentes, en el cual se muestran los datos de la guía a editar. Además del botón de guardado, se incluye un botón para eliminar la guía docente y, en la parte superior, un botón para regresar al listado de guías docentes.

- **Administración del buzón de sugerencias**

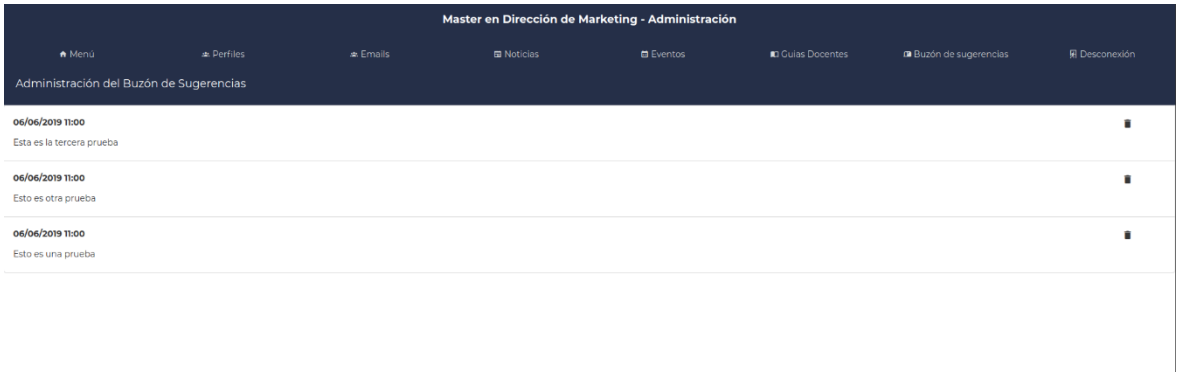

Figura 55: UI de la página de administración del buzón de sugerencias

La interfaz de la página de administración del buzón de sugerencias, solo visible para los coordinadores muestra un listado de las sugerencias enviadas al grupo del usuario. Por cada sugerencia, se muestra un boton de borrado para eliminar dicha sugerencia

- **Administración de grupos**

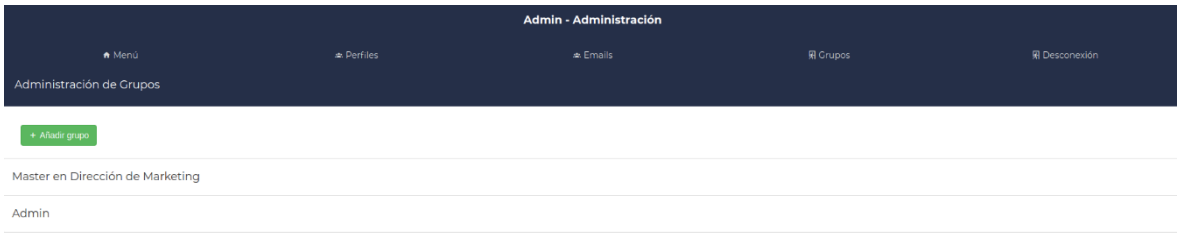

Figura 56: UI del inicio de sesión en administración

La interfaz de administración de grupos, solo visible para adminsitradores, muestra un listado de los grupos existentes en el sistema. Al pulsar el botón “Añadir grupo”, se despliega el siguiente formulario:

Figura 57: UI del formulario de creación de grupo

El usuario introducirá el nombre del grupo y pulsará guardar, apareciendo el nuevo grupo en el listado.